DESIGN AND REALIZATION OF AN EXPERIMENTAL OPTICAL STOP-MOTION CAPTURE SYSTEM

Bachelor Thesis at the Institute for
Media and Imaging Technology
Cologne University of Applied Sciences

Author: Sven Bambach
Mat.-Number: 11062047

First Reviewer: Prof. Dr. Stefan M. Grünvogel
Second Reviewer: Prof. Dr. Dietmar Kunz

Cologne, October 2010

Title: Design and Realization of an Experimental Optical Stop-Motion Capture System

Author: Sven Bambach

Reviewers: Prof. Dr. Stefan M. Grünvogel (Computer Animation and Computer Science)
Prof. Dr. Dietmar Kunz (Digital Image Processing and Mathematics)

Abstract: Motion capturing plays an important role in computer animation and combines many different fields of research in media engineering, such as camera calibration, marker detection via image processing and 3D reconstruction. Most professional motion capture systems are distributed proprietarily by a handful of companies and do not grant much insight into detailed workflows. The motivation for the project this thesis is based on was to autonomously develop a simpler, yet similar system, which would capture static poses of a puppet. This thesis points out the designing steps of this system and illustrates how many of the aforementioned aspects of motion capturing are answered using the example of the stop-motion capture system, thus building a basis for the comprehension of more complex systems.

Keywords: optical motion capture, marker recognition, camera calibration, 3D reconstruction

Date: 10/25/2010

Titel:	Entwurf und Umsetzung eines experimentellen, optischen Stop-Motion Capture Systems
Autor:	Sven Bambach
Referenten:	Prof. Dr. Stefan M. Grünvogel (Computeranimation und Datenverarbeitung) Prof. Dr. Dietmar Kunz (Digitale Bildverarbeitung und Mathematik)
Kurzbeschreibung:	Motion Capturing spielt eine tragende Rolle in der Computeranimation und verbindet viele Forschungsgebiete der Medientechnik, so wie Kamerakalibrierung, Marker-Erkennung durch Bildverarbeitung oder 3D Rekonstruktion. Die meisten professionellen Motion Capture Systeme werden proprietär von nur wenigen Firmen vertrieben und gewähren keine großen Einblicke in ihren detailierten Workflow. Die Motivation für das Projekt, auf dem diese Thesis basiert, lag darin eigenständig ein vereinfachtes, jedoch ähnliches System zu entwickeln, welches die statischen Posen einer Puppe erfasst. Diese Thesis erklärt die Designschritte dieses Systems und erläutert wie viele der oben erwähnten Aspekte des Motion Capturing am Beispiel des Stop-Motion Capture Systems beantwortet werden, um so eine Basis zum Verständis komplexerer Systeme herzustellen.
Stichwörter:	optisches Motion Capture, Marker-Erkennung, Kamerakalibrierung, 3D-Rekonstruktion
Datum:	25.10.2010

Contents

1 Introduction

The first part of the introduction deals with the motivation for this thesis and the project it is based on. The following part points out what should be accomplished by the thesis, while the last part explains its structure, as well as summarizes the following sections.

1.1 Motivation

Motion capturing plays an important role in computer animation in contrast to the popular misconception that it is only used in filmmaking or computer games. Research in general has a special interest in creating complex movements and realistic physical interactions, for instance for military or sport purposes or even medical applications.

Optical motion capturing in particular combines many different fields of research in media engineering. These can be essential matters such as finding the right illumination level for the capture scene, the development and calibration of appropriate cameras, the automatic analysis of video images, or the task of mapping the motion data onto a computer model.

Most professional motion capture systems are distributed proprietarily by a handful of companies and do not grant much insight into detailed workflows. The original motivation of the project that this thesis is based on was therefore to autonomously develop a similar system inspired by those professional systems.

After further considerations, it was decided to simplify this system in order to keep the project within a reasonable effort. The idea for a stop-motion capture system that would capture static poses instead of movements was developed. Many essential aspects and questions of motion capturing would be retained in this system and can thus be answered practically.

1.2 Goals

The project's goal can be summarized as the approach to experimentally designing a system that allows you to optically capture a puppet in order to three dimensionally reconstruct the puppet's pose. The build-up should be a convenient and portable size and the whole system should be easily installable onto any computer.

All designing decisions are to be reasoned and documented in this thesis. Consequently, the thesis is first and foremost supposed to serve as a practical introduction to the functional interaction of media and imaging technology such as image processing,

camera calibration or 3D reconstruction. Moreover, it is supposed to build a basis for the comprehension of more complex systems, such as professional motion capture systems.

Nonetheless, the data of the puppet's pose should not just be captured for the sake of capturing the pose itself, but to follow established computer animation standards that allow the system to have a practical usage as well. Possible fields of application could for instance include movie productions that rely on a mixture of stop-motion technology and computer animation (e.g. the 2009 fantasy film "Coraline" [IMD]).

1.3 Structure

The structure of this thesis primarily follows the single steps that were taken to realize the system in a chronological way. Every section covers a different problem and tries to describe and reason the different designing approaches that were required in regards to the occurring problems. As needed, the problem solving is either pointed out generally or specifically for this system.

Section 2, "Preparations", deals with general considerations about the hardware build-up of the stop-motion capture system, as well as the question of finding an appropriate programming language with the necessary frameworks/libraries to develop the system's software.

Section 3, "Locating and Identifying the Markers", includes all questions concerning the puppet's optical markers. Starting with explaining basic requirements that the markers need to fulfill, it points out how and where the markers are to be attached on the puppet and lastly describes all image processing steps that are needed to automatically locate and identify the markers within a camera image.

Section 4, "Calibrating the Camera", first explains the principles of camera calibration by explaining the single steps that are needed to formulate a mathematical correlation between objects within the scene and within the camera image. On this basis, the section also points out how the calibration parameters for the system's webcam were practically estimated.

Section 5, "Reconstructing the Marker Positions", describes how one can estimate the puppet's markers' positions in space based on the camera images and with the help of the parameters estimated in section 4.

Finally, section 6, "Reconstructing the Puppet Pose", answers the question how to define a skeleton for the puppet that helps to reconstruct the puppet's three dimensional pose based on the its markers.

Unlike the previous sections, section 7, "About the Software", does not refer to a

specific problem, but introduces the software "JumpingJack 3D Capturer", that was developed for this stop-motion capture system. This software serves as a user interface that enables one to operate the entire system and implements all algorithms and calculations that were introduced in the previous sections.

Section 8, "Conclusion", considers the results of this project and critically remarks about which aspects could be the subject of further improvement.

2 Preparations

The next necessary step, after the idea had been formulated, was to make certain preparations as well as the first designing decisions. The first part of this section points out considerations about the hardware build-up that would lead up the technical drawing. The second part describes the software considerations for the system.

2.1 Hardware Considerations

When planning the build-up, it was inspirational to draw on the insight of existing motion capturing systems wherever possible. Afterwards the build-up was simplified in order to keep the system as concise and reliable as possible. In Figure 1, a person, equipped with markers, is shown in a classic motion capturing scenario. The person is standing in the center of a series of cameras spaced evenly in a circular formation and directed towards the center. Even though the theory states that it is basically enough to have two differently positioned cameras capturing the same point, or in this case marker, in order to reconstruct its position (see section 5), considerably more cameras are used. The simple reason for this is that some markers can be blocked from view depending on the movement or pose of the person at a specific point in time. By using multiple cameras, one tries to ensure that every marker is captured by at least two cameras at each point in time.

Figure 1: *VICON F40 Motion Capture System* [Rel09]

The system created in this project faces the same problem. Nonetheless it is possible to significantly simplify the build-up at this point. Unlike actual motion capturing, it is not necessary to reconstruct movements, just a static pose. Therefore one can afford to

take the single pictures of each camera sequentially, so that the build-up only requires one camera. However, changing that camera's position and orientation in between every picture is both exhausting and prone to error because both parameters must be known precisely for every picture taken. It makes much more sense to install the camera at a fixed position and move the puppet relatively to the camera. As long as this is kept in mind when defining the coordinate systems (see section 4.2.1), it makes practically no difference. For instance, if the puppet is turned around on its own axis on a defined position, with a camera located in the distance r from this position, this has the same effect as if the camera was moved along a circular path with the radius r around the puppet.

2.1.1 Choosing the Camera

The objective was to have portable build-up and a system that could be easily installable on any computer. The camera should therefore use a standardized interface that is supported by most computers and operating systems. A simple USB webcam fulfills this requirement. Another advantage of using a webcam is that a series of webcam models is equipped with light emitting diodes to light up the scene in order to gain relatively low-noise pictures, even in dark surroundings. This is a useful aspect because controlled lighting is an important factor when recognizing the markers. This will be explained in detail in section 3.1 and 3.2.

For the actual camera the model "Deluxe Optical Glass" by "Hercules" was chosen. The camera is equipped with four white light emitting diodes that are distributed around the lens. All components are implemented into a solid metal body that one can unscrew from its mounting clamp. Due to this feature it was easy to install into the build-up.

2.1.2 Choosing the Puppet

The choice of the puppet is determined by the goal of creating the widest possible range of poses. For the puppet this means to have as many jointed limbs with as many degrees of freedom as possible. A lay figure, that is normally used as a reference for human anatomy by artists, qualifies perfectly for this. In addition to this, a lay figure's surface is smooth and without disruptive details that would make it difficult to attach markers.

The figure that was chosen has its pelvis attached to a metal axis that is resting on a circular platform. This has the advantage that one can perfectly use this axis as the turning axis for the figure, but comes at the price that the pelvis's position and

orientation is fixed. This figure is available in different heights (210mm and 310mm). The decision on the lay figure's size will be explained in the following section.

2.1.3 Planning the Build-Up

Before the final dimensions of the build-up could be determined, it was necessary to decide on the puppet's size. The camera is supposed to capture the puppet with its limbs fully outstretched at full frame. At the same time, the distance from the puppet to the camera should not be too long in order to keep the whole build-up at a convenient size.

To estimate the puppet's maximum height, the following parameters were taken as a basis: The camera's focal length is specified as $f = 4.22mm$. The size of its C-MOS Chip is 1/4". This conforms to a sensor height of $B = 2.44mm$ [Sta]. Notice here that the actual active diameter of the sensor is only about two thirds of the denoted diameter (1/4"). 50cm was considered to be appropriate as the maximum distance, g, from the puppet to the camera. According to Paul A. Tipler [Tip07, page 1050 - 1055], when approximating the camera as a thin converging lens:

$$\frac{1}{f} = \frac{1}{g} + \frac{1}{b} \tag{2.1}$$

$$\frac{B}{G} = \frac{b}{g} \tag{2.2}$$

Therefore, G can be estimated as:

$$G = gB(\frac{1}{f} - \frac{1}{g}) \tag{2.3}$$

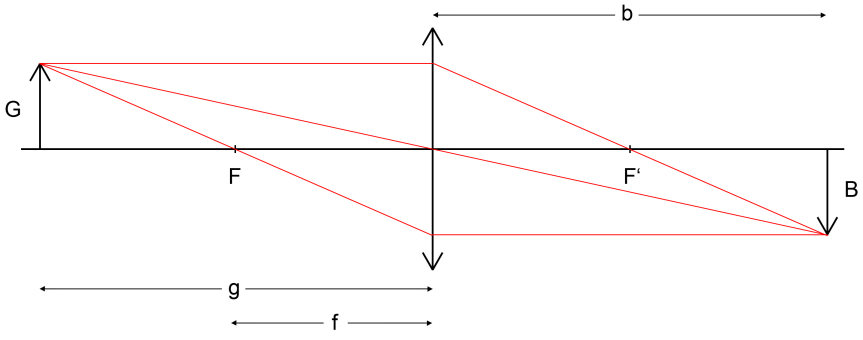

Figure 2: *Ray Tracing Diagram for a Thin Converging Lens*

The parameters involved in these equations are defined in figure 2. According to this, the maximum height of the puppet is $G = 229mm$. Therefore, the 210mm lay figure was found to be the best choise.

The actual measurements of the whole build-up were determined practically using a rough-and-ready build-up. The drafted out mechanical drawing that was handed to the university's fine mechanical workshop can be seen in figure 3. The most important distances in order to precisely determine the position of the camera are the distance from the center of the lens to the surface of the bottom panel (80mm) as well as the distance from the puppet's rotation axis to the lens (400mm).

The camera and the puppet are positioned along one optical axis. The puppet's platform is embedded into the bottom panel and can be turned around by 360°. The back panel ensures that the camera's view has a plain background. The material for the planes is simple wood.

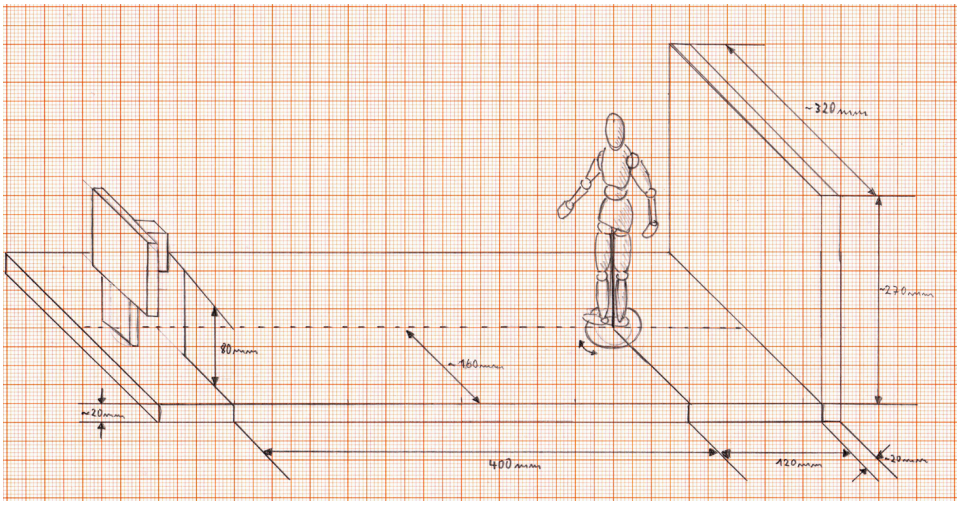

Figure 3: *Technical Drawing of the Build-Up*

2.2 Software Considerations

When planning a project that requires a substantial amount of software development, the first decision that needs to be made is choosing an appropriate programming language. This decision was based on the following requirements: Having a way to simply implement a graphical user interface and the existence of well documented, preferably free libraries/frameworks, that allow the following three steps to be realized:

- Recognizing and capturing a webcam with the option of grabbing certain frames

as images files

- Easily managing and processing those images
- Rendering a three dimensional scene to display the figure's pose

Java offers appropriate, well documented frameworks for all those requirements. The Java Media Framework was chosen for streaming the webcam, ImageJ was chosen for processing the images and JMonkey was chosen as a 3D Engine.

The following sections shall quickly introduce those three frameworks and describe how they offer solutions for the project.

2.2.1 Java Media Framework (JMF)

The Java Media Framework [Ora] is a time-based media processing framework developed by Sun directly. It is an extension of Java SDK, containing the following main functions:

- Real-time processing of media data
- Capturing media data streams
- Saving media data
- Streaming and transcoding media data

JMF was last updated on November 2004 because Sun, from that time on, focused their efforts in the region of multimedia on JavaFX, a framework for developing rich internet applications. Nonetheless, JMF's API is quite sufficient for the purpose of the streaming the webcam. The following shall outline how to capture a Webcam using JMF.

The API offers a *CaptureDeviceManager* that returns a list of all available capture devices. That list can be filtered by a certain *Format*, for example only to return devices that support a certain video format. Having the desired capture device, a *DataSource* can be created, from a *Processor* object can finally be instantiated. The Processor object provides all necessary methods to start and stop a stream, as well as methods to return the display component, which for instance can be built into a "Swing" interface to display the video stream. A more detailed explanation of how to use JMF to capture media data streams can be found in "Medienverarbeitung in Java" [Eid04].

2.2.2 ImageJ

ImageJ [oH] is a free Java-based image processing program developed by the National Institutes of Health. It offers various standard image processing functions and has an open architecture that allows one to easily write plug-ins. Its architecture also allows one to use ImageJ as a library.

The following shall outline how to process an image in Java using the ImageJ API:

An image is stored in an *ImagePlus* Object that can return an *ImageProcessor*. There are classes extending this *ImageProcessor* for each image type, for instance a *ByteProcessor* for an 8 bit grayscale image or a *ColorProcessor* for a 32 bit RGB image. While each processor already provides a selection of matching processing methods such as convolving, cropping etc., the *getPixels()* Method simply returns all pixel values as arrays of the corresponding data type, allowing you to read and manipulate each pixel value at will.

ImageJ also provides the Java interface *PlugInFilter* that allows one to easily write plug-ins for ImageJ using the methods described above. The Hough Transform described in section 3.3.2 was implemented as such a plug-in.

2.2.3 jMonkey Engine (jME)

The jMonkey Engine framework [jMo] is a community-driven, open-source graphics API for Java, based on 3D scene graphs. jMonkey itself uses the Open Graphics Library (OpenGL), developed by the Khronous Group, tfo display 3D scenes in real-time.

jMonkey easily allows various objects, from simple geometrical bodies up to complicated meshes, as well as different sources of light (e.g. point light), to be organized in a scene graph. It also allows one to define and change camera parameters and will render the final image. jMonkey offers the Java interface *SimpleGame* that provides all necessary methods to create such a scene.

Fortunately, jME also implements a skeletal system, that is intended to animate characters and is therefore perfectly suited for the system's purpose of reconstructing the pose of the stop-motion puppet. The use of this skeletal system for the project is described in section 6.

3 Locating and Identifying the Markers

This section summarizes all steps that were taken in order to enable the stop-motion capture system to locate and identify the optical markers on the puppet. The first part outlines general considerations about the marker qualities, and about how to usefully attach them onto the puppet. The second part describes how the different marker colors were chosen and measured. The last part explains the image processing techniques that are needed in order to identify the markers on the camera's image.

3.1 Marker Considerations

The system is supposed to be capable of reconstructing the pose of the puppet. Therefore, it must optically capture the puppet. Since some of the puppet's qualities are already known (namely its body structure and the dimensions of its limbs) one doesn't need to capture it as a whole. In fact, it is sufficient to capture discrete points on the puppet's body and associate them with the corresponding body parts. Those points are highlighted by optical markers.

The following two sections point out how to ensure that those markers are, first of all, recognizable and secondly, identifiable (so that they can be associated with a body part), for the system. Lastly, section 3.1.3 answers the question of how the markers are to be placed on the puppet.

3.1.1 Requirements to Locate a Marker

When looking back at 1, one notices that the markers on the person are shining. Actually, they are reflecting the flash of the photo camera. During the motion capturing, these markers are shined on by light emitting diodes from the mocap cameras. These diodes emit infrared light, light with wavelengths below the visible spectrum, that is reflected by the markers. Appropriate filters in the camera's lens then filter out all light beyond the infrared range, thus keeping all ambient light out of the camera's view, making the markers appear to be bright. Figure 4 shows two of those mocap cameras by the company Optitrack.

The build-up of the stop-motion capture system does not quite have those capabilities. However, the webcam, as described in section 2.1.1, is equipped with four white light emitting diodes, with which the scene can be directly illuminated. Furthermore, the back panel and the bottom panel make sure that the scene has a plain background. By painting those panels in a flat black and ensuring that the camera LEDs, that illuminate

the puppet from a front angle, are the scene's dominant source of light, the end result is a clearly illuminated puppet. By also painting the puppet's body in the same flat black, one can highlight the markers in the same way. Of course this only works when the markers' reflectance coefficients, respectively their remission's coefficients, are sufficiently high. Section 3.2 deals with this issue.

Figure 4: *OptiTrack S250e (left) and OptiTrack Flex:V100R2* [Opta]

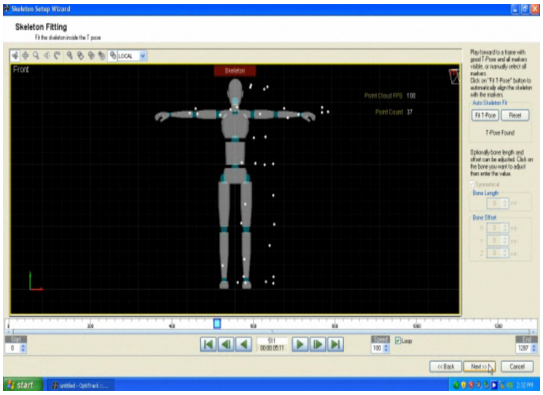

Figure 5: *T-Pose Calibration Screen, ARENA Motion Capture Software* [Optb]

3.1.2 Requirements to Identify a Marker

In regards to the identification of a marker, the stop-motion capture system is forced to follow a significantly different approach than most motion capture systems. Those systems usually go back on the so called "T-Pose Calibration". For that calibration, the actor, after being equipped with the markers and before the actual capturing starts, needs to go into a defined pose (mostly the "T-Pose"). Using a graphical interface and special software, the markers are then identified based on their position in this pose and can then be associated with the body parts. Figure 5 illustrates such a calibration. From this time on, the markers are tracked from their initial positions.

This kind of calibration makes no sense for the stop-motion capture system because it must exclusively work with static poses. Therefore the markers need an intrinsic quality that makes them instantly identifiable in the scene. Such qualities could, for instance, be different shapes or colors. However, multiple shapes would lead to more complex image processing. The impact of different shapes would be even more substantial considering that all shapes that differ from a sphere will have different outlines on the projected image of the scene. This depends on the point of view of the camera, as well as on the

marker's orientation. It makes much more sense to differentiate the markers based on their color. Sections 3.2 and 3.3.3 describe in detail how this was done for this project.

3.1.3 Placement and Quantity

The following deals with the question of how to usefully place the markers onto the puppet, in order to reconstruct the puppet's pose based on the markers' positions. As previously mentioned in the beginning of this section, the dimensions of the single limbs are already known. Therefore, as section 6 will show in detail, it is sufficient to determine the limbs' orientations. One possible way to accomplish this is with the placement of two markers on each limb, as described in figure 6. If the upper marker has the position m_1 and the lower marker has the position m_2:

$$\mathbf{a} = m_2 - m_1 \tag{3.1}$$

$$\mathbf{a_n} = \frac{\mathbf{a}}{\|\mathbf{a}\|} \tag{3.2}$$

This provides a directional vector $\mathbf{a_n}$ that points in the exact same direction as the lower arm. Of course this assumes the markers' alignment towards each other is parallel to the orientation of the corresponding limb, as it is the case in figure 6.

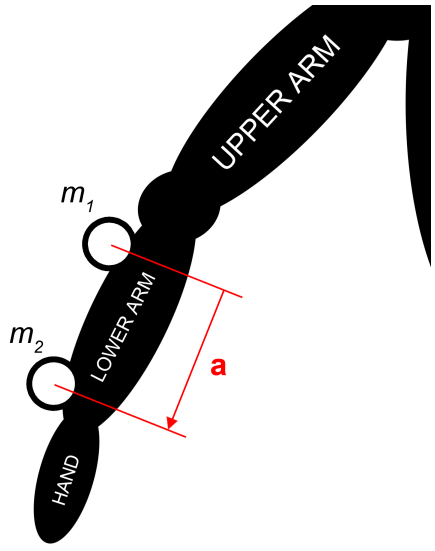

Figure 6: *Estimating the Lower Arm's Orientation*

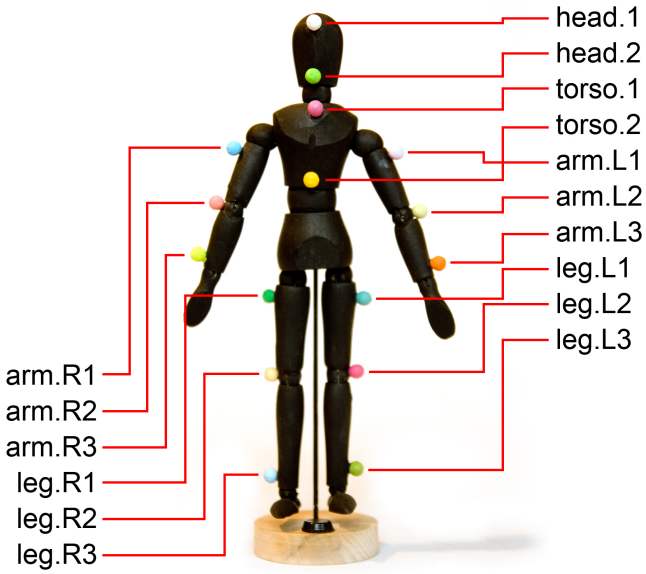

Figure 7: *The Painted Black Lay Figure Equipped with All Markers*

Figure 7 shows the lay figure that was used for the project, after it was equipped with

markers based on the concept described above. As original material for the markers, plastic beads with a 6mm diameter were used. In order to minimize the marker quantity for both arms and legs, one marker each was placed on the elbow joints and knee joints respectively. In this way for instance, the marker on the left elbow joint serves as the lower marker for the left upper arm, as well as the upper marker for the left lower arm. With all the markers on the puppet, it is therefore possible to determine the orientation of both left and right upper arm and lower arm, both left and right upper leg and lower leg, as well as the torso and the head.

The pelvis does not have markers because it's fixed and cannot rotate. Both hands and feet were not equipped with markers either, for reasons of practicality.

3.2 Color Calibration

The decision of identifying the markers by their color being made, this section describes how the color values for the markers were practically determined, not as a general approach, but for this specific project.

As previously mentioned, the original material for the markers consisted of white, glossy plastic beads that needed to be painted individually. For the choice of the colors, the following two assumptions were considered: The bead color should be as uniform and plain as possible so that it could later be extensively measured in the image. Therefore, it was appropriate to work with flat colors, because glossy colors could create unwanted specular reflections on the bead. Furthermore, as suggested in section 3.1.1, it was appropriate to chose colors with high remission's coefficients. This means using bright colors, so that the markers are recognizable in the scene.

A first set of beads (henceforth just referred to as "markers"), painted with acrylic paint, was evenly distributed in the scene instead of the puppet. Afterwards, with exclusive illumination of the webcam's LEDs, a picture of the scene was taken. In this picture, all markers that appeared to be too dark to be clearly recognizable as a circular shape to the naked eye, were excluded from further considerations. Using the image processing software "Adobe Photoshop CS4", the color values of all remaining markers was measured in HSB (hue, saturation, brightness) coordinates. The color value was thereby estimated as the arithmetic average of all pixel values in a 5×5 pixels square around the marker's center. With an average marker circle size of $5\,pixels < r < 7\,pixels$, this conforms to the biggest possible plain area within the circle that was not influenced by darkened pixels by the circle's edge. Figure 8 illustrates this.

This procedure was successively repeated with newly mixed colors until 16 markers

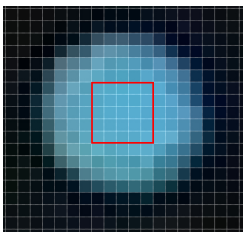

Figure 8: *Measuring the Color Value of a Marker*

with color values, which were spread apart from each other within the HSB color space, were found. Thereby, it needed to be taken into consideration that the hue values in the HSB color space are arranged periodically. This means that, on a scale from 0 to 1, hue 0.999 is right next to hue 0.001.

However, the color values for the 16 markers, as they were measured above, were not properly qualified as final reference values that one could use to properly identify the markers within an image. Those reference values were determined as following: To keep the measuring error at a minimum, the markers were distributed onto the puppet, as described in section 3.1.3, and the figure was arranged to a "T-Pose". After that, with exclusive illumination of the webcam's LEDs, twelve pictures (from every angle at 30° steps) were taken from the puppet. In all twelve pictures, all color values of all markers (as long as they weren't blocked from view) were measured. The reference color value for a marker was estimated as the arithmetic average of all measured color values.

The following table lists up the color values determined for the markers that were used for the project. The markers' names can be found in Figure 7.

	H	S	B		H	S	B
head.1	0.499	0.176	0.754	**arm.R2**	0.978	0.265	0.598
head.2	0.303	0.611	0.607	**arm.R3**	0.240	0.619	0.675
torso.1	0.768	0.329	0.562	**leg.L1**	0.514	0.540	0.599
torso.2	0.187	0.633	0.706	**leg.L2**	0.867	0.387	0.575
arm.L1	0.589	0.234	0.804	**leg.L3**	0.286	0.505	0.515
arm.L2	0.286	0.251	0.694	**leg.R1**	0.387	0.667	0.558
arm.L3	0.077	0.715	0.575	**leg.R2**	0.188	0.237	0.777
arm.R1	0.558	0.605	0.827	**leg.R3**	0.554	0.431	0.790

3.3 Image Processing

The following shall first explain how the system can automatically determine the position of a marker within an image using image processing. This is achieved by a Hough Transform, which will be explained in section 3.3.2. In order for the Hough Transform to work, one needs an edge map to begin with. How an edge map can be created based on the original image shall be explained in section 3.3.1.

With the markers' positions having been found within the image, the markers finally need to automatically be identified by their colors. This process is pointed out in section 3.3.3.

3.3.1 Creating an Edge Map

The first step in the image processing flow is the creation of an edge map. An edge map is a binary image in which the edges that were found in the original image are marked in white, while the rest is marked in black. An edge within an image can thereby be explained with the following definition: "Edges can roughly be explained as those places within an image, where its intensity changes dominantly within a small area and along a distinctive direction", Wilhelm Burger [Bur06, page 117].

There are a range of different filters that are usually used for detecting edges. Most of them are based on estimating the local gradient vector of the image function, meaning the derivative of the image's intensity function along both image directions. A famous variation of such a filter is the so called Prewitt Operator [Bur06, page 578 - 579]. This is mentioned here for the sake of completeness, but will not be explained in detail because a completely different approach for creating the edge map was used in this project. This approach takes advantage of the specific qualities that all images of the scene have in common and creates an edge map in three simple steps. These steps are explained in the following:

The original image is shown in figure 9 (a). It can be characterized by its dark, near black background and relatively bright markers in the image foreground. Thus, by finding an appropriate threshold value, one can easily reduce the image to a binary image (for instance a black background with white markers) using a simple threshold operation. In order to do this, the image first needs to be reduced to its brightness information. This can be done by a simple conversion from the original 24bit RGB image to an 8bit grayscale image, which is shown in figure 9 (b). Next, one must find an appropriate threshold brightness. ImageJ already implements a method to automatically find such a

value. It estimates the threshold by taking a test threshold and computing the average of the pixels at or below the threshold and above. It then computes the average of those two, increments the threshold, and repeats the process. Incrementing stops when the threshold is larger than the composite average. The binary image that results after such a threshold operation can be seen in figure 9 (c).

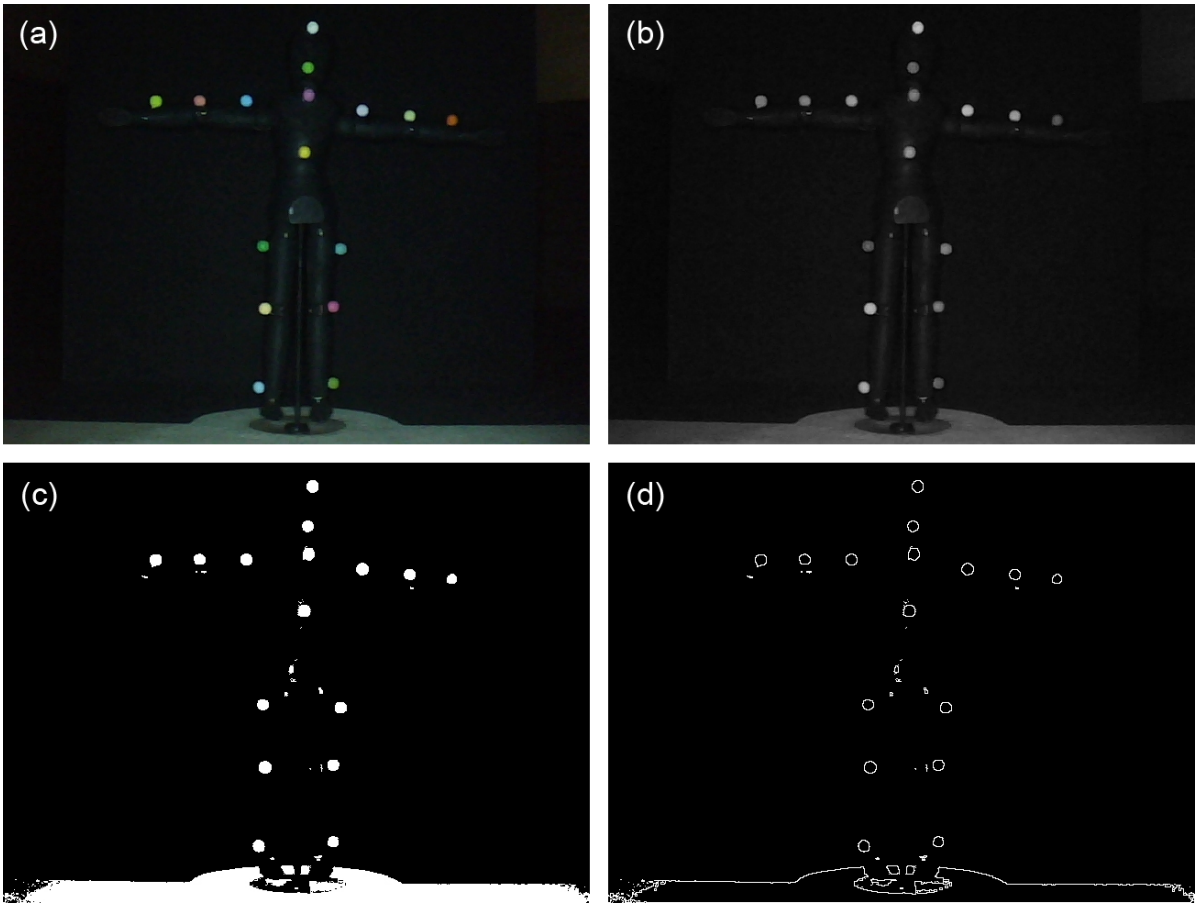

Figure 9: *(a) Original 24bit RGB Image, (b) 8bit Grayscale Image, (c) Binary Image After Threshold Operation, (d) Edge Map After Outlining*

In order to obtain the desired edge map for the Hough Transform, the white circles of the markers need to be outlined. This can be done using a simple binary filter with a 3×3 kernel that slides along each pixel of the image. If the current pixel is black, it will stay black. If the current pixel is white, it will become black only if all eight neighboring pixels are white. If not, the pixel will stay white. Figure 10 illustrates this. The resulting edge map can be seen in figure 9 (d).

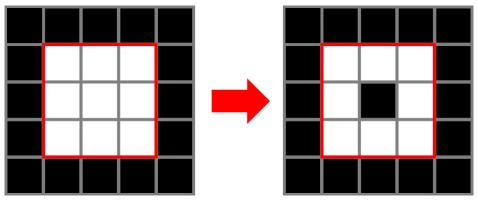

Figure 10: *Example of the Binary Filter That Is Used to Outline the Binary Image*

3.3.2 Hough Transform

The positions of the circles within the edge map can now be determined using a Hough Transform. This method was originally published by Paul Hough as a US patent in 1962 and offers a general approach to locate any pattern within an image, as long as it can be parameterized (for instance lines or circles). The following shall outline the basic principle of the Hough Transform, first with the example of recognizing a straight line as a pattern. This explanation has been partly taken from the book "Digitale Bildverarbeitung" [Bur06, page 156 - 169]:

A straight line within the xy-plane can be described with two parameters:

$$y = kx + d \tag{3.3}$$

In this, k is the line's slope and d its point of intersection with the y-axis. A straight line that goes through two given (edge-) points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, must therefore fulfill the following two equations:

$$y_1 = kx_1 + d \text{ and } y_2 = kx_2 + d \tag{3.4}$$

The goal is now to find those lines that have the most edge points on them. In order to accomplish this, one could simply determine all possible straight lines within an image and count the amount of edge points on each one. However, this would be inefficient due to the large amount of possible straight lines. The Hough Transform looks at the problem the other way around, by first looking at all possible straight lines, L_j, that go through a given point $p_0 = (x_0, y_0)$:

$$L_j : y_0 = k_j x_0 + d_j \tag{3.5}$$

When considering k_j ad d_j as variables and x_0 and y_0 as the constant parameters:

$$M_i : d = -x_i k + y_i \tag{3.6}$$

In this equation, M_i is also a straight line, but it lies within the parameter space (also called a Hough space) where the axes are k and d. The relation between the image space and the parameter space can be summarized as:

image space (x, y)	parameter space (k, d)
point $p_i = (x_i, y_i)$	$M_i : d = -x_i k + y_i$ line
line $L_j : y = k_j x + d_j$	$q_j = (k_j, d_j)$ point

Every point p_i within the image space corresponds with exactly one line M_i within the parameter space. The positions within the parameter space where multiple lines intersect are the topic of interest. As outlined in figure 11, line M_1 and line M_2 intersect at the position $q_{12} = (k_{12}, d_{12})$ in the parameter space. This position conforms to exactly that line within the image space, which goes through point p_1, as well as point p_2. Generally speaking, this means: If N lines intersect at a position (k', d') within the parameter space, then N points lie on the corresponding line $y = k'x + d'$ within the image space.

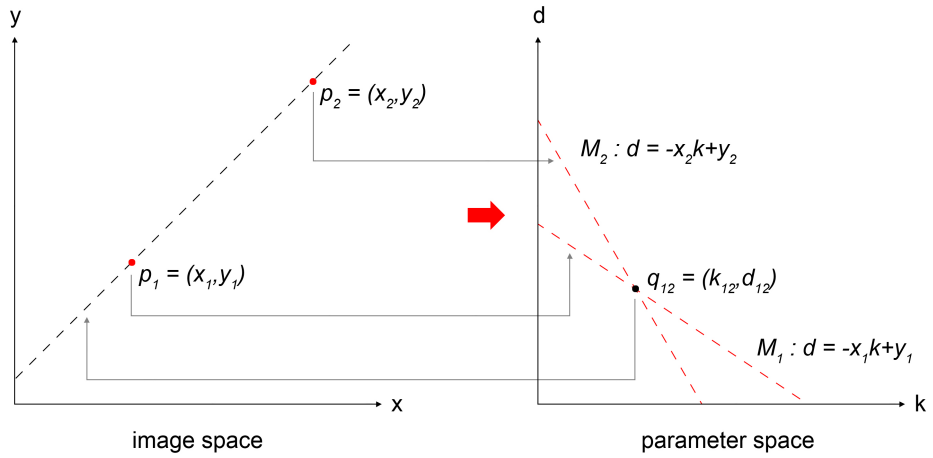

Figure 11: *Correspondence Between the Image Space and the Parameter Space*

Therefore, finding dominant lines in an image is equal to finding those positions within the parameter space where as many lines as possible intersect. This is precisely the idea behind the Hough Transform algorithm. Figure 12 shows an image with discrete pixels and the corresponding discrete parameter space. This image could, for example, be a

cropped out part of the edge map that was created in section 3.3.1. The two lines for both image points are cumulatively written into the parameter space. The "pixel" with the counter reading "two" conforms to the line that goes through both image points.

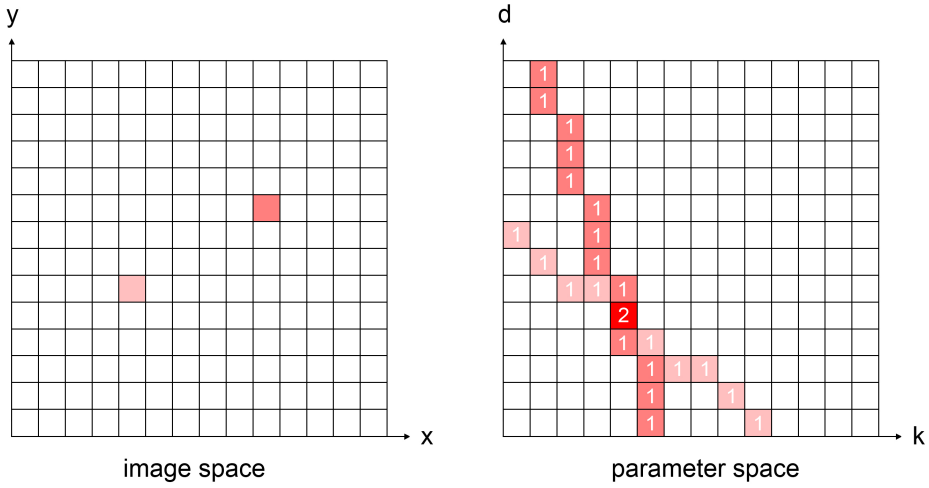

Figure 12: *Image Space and Parameter Space for an Image with Discrete Pixels*

This project's system is not interested in finding lines, but in finding circles. A circle can be described with three parameters, its center point's coordinates $p = (x, y)$ and its radius r. Thus, one needs a three dimensional parameter space in order to find any desired circles within an image. Every point on a circle with the radius r_i within an image corresponds with a circle within the $[x, y, r_i]$ plane of the parameter space.

The implementation of the Hough Transform was done using the ImageJ plug-in "Hough Circles", which is available on the ImageJ website [Pis09]. As parameters, the plug-in receives the minimum radius and maximum radius of the circle, as well as the maximum number of circles one desires to locate. For instance, when looking for N circles with a radius $r_{min} < r < r_{max}$, the centers of those N circles are returned that contain the most image points on their circular paths from r_{min} to r_{max}. As standard values, the system looks for 16 circles (conforming to 16 markers) with a radius $5pixels < r < 7pixels$. If desired, the parameters are adjustable for every image. Figure 13 (a) shows the center points of the markers that were recognized, and then displayed as white crosses.

The exact implementation of the Hough Transform can be found in the *HoughCircles.java* class (See section 7.2 for more details on the software architecture).

3.3.3 Color Recognition

The color recognition forms the last step in the image processing flow. After the Hough Transform, the center points of the potential markers within the image coordinate system are known. Based on those coordinates, the color values of the possible markers can be measured in the original 24Bit RGB image. As described in section 3.2, the color value is measured as the average of all pixel values in a 5×5 pixel square around the center point. Afterwards, the color distance Δ_{color} from all potential markers to all 14 color references is determined with the following formula:

$$\Delta_{color} = \sqrt{\Delta^2_{hue} + \Delta^2_{saturation} + \Delta^2_{brightness}} \tag{3.7}$$

After that, all delta values are ordered increasingly from smallest to largest. It is then checked successively, as to whether the color distance from the potential markers to the reference colors is smaller than a certain value Δ_{max}. When this is the case, the corresponding marker counts as identified, as long as it has not already been identified. Ordering the deltas thereby ensures that only those markers with the smallest possible color distance are chosen. The Δ_{max} value ensures that foreign structures that have falsely been located by the Hough Transform will not be recognized as markers. The exact implementation of the color recognition algorithm can be found in the *ColorIndentifier.java* class (See section 7.2 for more details on the software architecture).

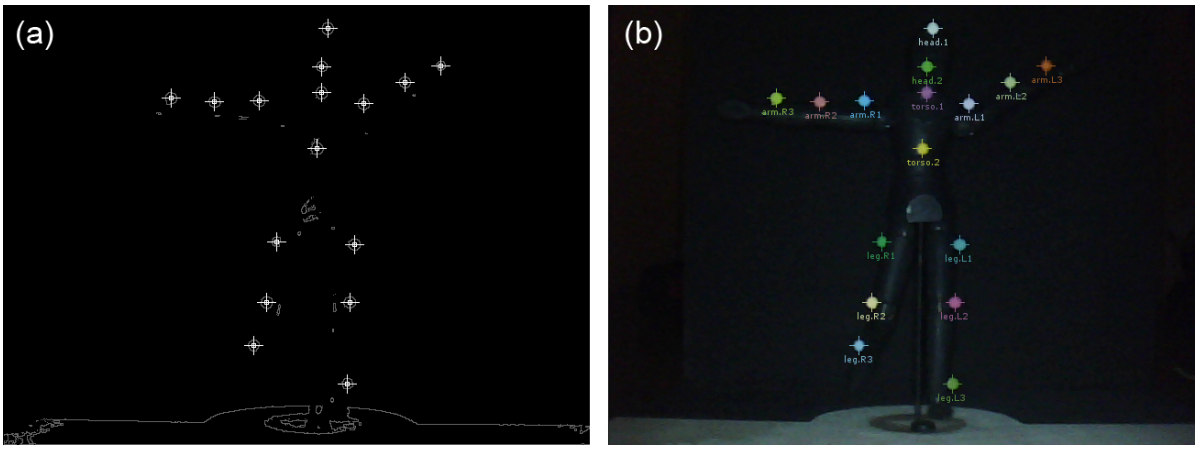

Figure 13: *(a) Located Markers Within the Edge Map, (b) Identified Markers Within the Original Image*

4 Calibrating the Camera

The positions of the markers within the image(s) are now known. The next goal of the stop-motion capture system is to use that information in order to reconstruct their positions within the scene. Generally speaking, this means determining a scene point based on a series of image points. To accomplish this, one first needs to be able to go the reverse way and find the relation between a scene point and the resulting image point. This means being able to predict where a certain point within in the scene will be projected in the camera's projection pane.

In order to do so, a series of parameters that describe the camera's behavior need to be estimated. This process is called camera calibration.

The parameters can be distinguished between intrinsic and extrinsic parameters. The first part of this section explains the meaning of the intrinsic parameters and points out how they were determined. The second part does the same for the extrinsic parameters, while also introducing the world coordinate system that was used for the build-up.

4.1 Intrinsic Parameters

The intrinsic parameters describe what can be considered as the inner behavior of the camera. Therefore, they are independent of the camera's position and orientation. The parameters' purpose is to describe the relation between the camera coordinate system and the image coordinate system. To determine those parameters, one can make use of the pinhole camera model. The following points out this approach in detail.

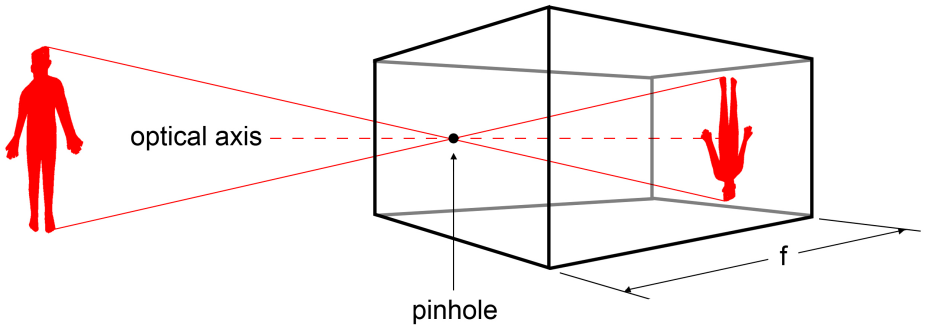

Figure 14: *Projection of a Pinhole Camera*

4.1.1 The Pinhole Camera and the Projection Matrix

A way to easily describe the projective qualities of a camera is by modeling it as a pinhole camera, which is illustrated in figure 14. The diameter of the pinhole lens is infinitely small and therefore serves as a perfect optical center. All rays that emanate from the object concentrate within this optical center and create an undistorted image on the projective plane. This can be considered as a perfect perspective projection.

Figure 15 now describes the system's camera based on this pinhole model. The optical center C of the camera is the equivalent of the pinhole lens where all the rays are concentrated. The z-axis points in the viewing direction of the camera. The projection pane is perpendicular to the z-axis at $z = f$, where f is the focal length of the camera. When, for example, f is decreased, the projective plane approaches the optical center thus making the camera angle wider.

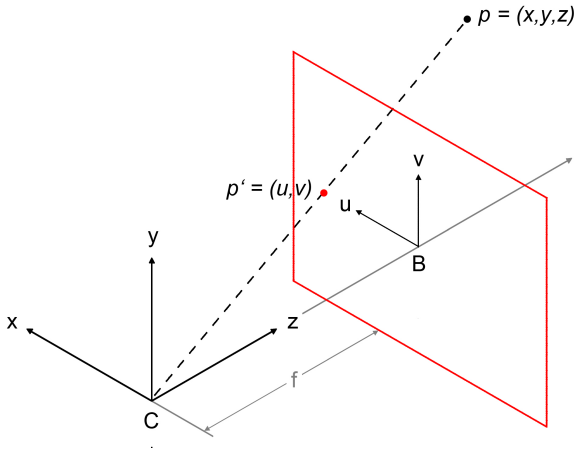

Figure 15: *The Camera as a Pinhole Model*

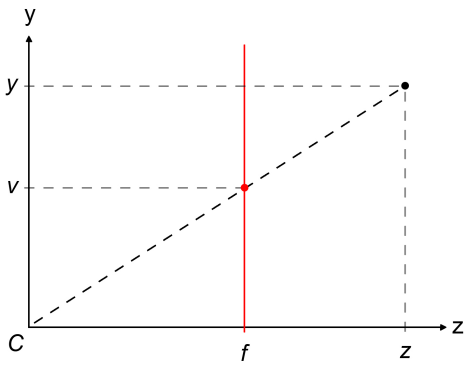

Figure 16: *Side View of the Camera*

As one can also see in Figure 15, point $p = (x, y, z)$ is projected onto point $p' = (u, v)$. The goal is now to express this transformation from a point p within the coordinate system (C, x, y, z) of the camera to a point p' within the coordinate system (B, u, v) on the projective plane. Based on the side view of the camera, as shown in figure 16, one can see that:

$$\frac{v}{f} = \frac{y}{z} \tag{4.1}$$

$$\Rightarrow v = f\frac{y}{z} \tag{4.2}$$

Similar to this:

$$\frac{u}{f} = \frac{x}{z} \tag{4.3}$$

$$\Rightarrow u = f\frac{x}{z} \tag{4.4}$$

This can be expressed with the following projection matrix P_{pro}:

$$\begin{pmatrix} U \\ V \\ S \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{4.5}$$

$$u = \frac{U}{S}, v = \frac{V}{S}, S \neq 0$$

The usage of homogeneous coordinates in equation 4.5 also shows that all points $p_r = (rx, ry, rz), r \in \mathbb{R}$, are projected onto the same point p'.

4.1.2 The Camera Coordinate System and the Image Coordinate System

The following deals with the process of dimensioning and adjusting the camera coordinate system, as well as the image coordinate system within the projective plane:

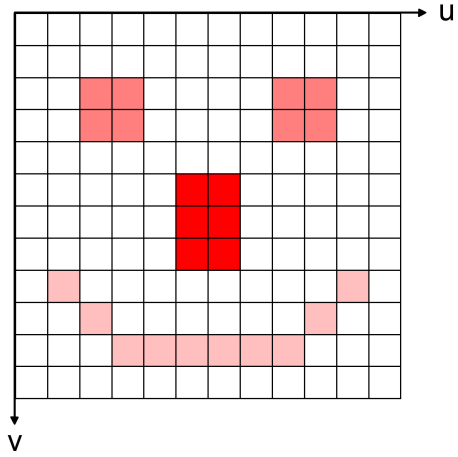

Figure 17: *The Image Coordinate System*

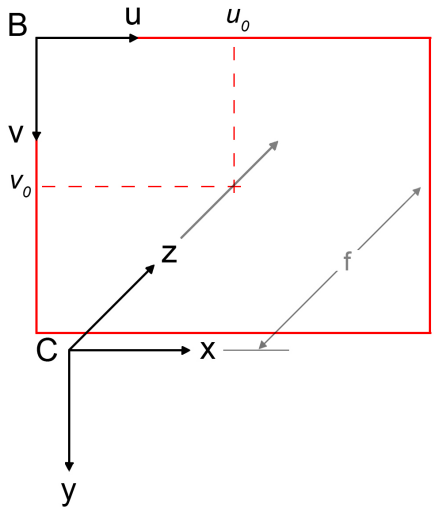

Figure 18: *The Adjusted Pinhole Camera*

In practice, the coordinates of a point within the projective plane are expressed as

image coordinates. The image coordinate system has its origin in the top left corner of the image. The u-axis is pointing to the right while the v-axis is pointing down. This is shown in figure 17. Therefore, the actual center of the image coordinate system is not the z-axis's point of intersection with the projective plane, as it was assumed in figure 15. To simplify matters, one now turns the camera coordinate system by 180° around its z-axis, causing the x-axis to point into the same direction as the u-axis and the y-axis to point into the same direction as the v-axis. By doing so, the whole transformation can be described by a simple translation by u_0 and v_0. Figure 18 clarifies this. Therefore, equations 4.2 and 4.4 become:

$$v = f\frac{y}{z} + v_0 \tag{4.6}$$

$$u = f\frac{x}{z} + u_0 \tag{4.7}$$

Furthermore, the different dimensions of the two coordinate systems need to be taken into consideration. While the quantities x, y and z are expressed in units of length (meters, for example), u and v are expressed in pixel units. Taking that into account with an appropriate scaling factor, equations 4.6 and 4.7 become:

$$v = fk_v\frac{y}{z} + v_0 \text{ with } k_v = [pixel/m] \tag{4.8}$$

$$u = fk_u\frac{x}{z} + u_0 \text{ with } k_u = [pixel/m] \tag{4.9}$$

Under the assumption that the pixels have square shapes, it is $k_u = k_v = k$. With $kf = c_{scale}$, it is:

$$v = c_{scale}\frac{y}{z} + v_0 \tag{4.10}$$

$$u = c_{scale}\frac{x}{z} + u_0 \tag{4.11}$$

Considering this, the projection matrix P_{pro} from 4.5 is now:

$$\begin{pmatrix} U \\ V \\ S \end{pmatrix} = \begin{pmatrix} c_{scale} & 0 & u_0 & 0 \\ 0 & c_{scale} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{4.12}$$

$$u = \frac{U}{S}, v = \frac{V}{S}, S \neq 0$$

This projection matrix now describes onto what point (or pixel) within the image coordinate system a point p within the camera coordinate system is projected. The intrinsic parameters can thereby be summarized as:

f $[m]$	effective focal length
k $[pixel/m]$	scaling factor
u_0 $[pixel]$	u-value of the image focal point
v_0 $[pixel]$	v-value of the image focal point

4.1.3 Estimating the Intrinsic Parameters

The most common approach to estimating the intrinsic parameters of a camera is based on determining a series of point pairs, each containing a point in the scene and the corresponding point in the image. To do so, one uses a calibration object with known dimensions and a series of defined points and then detects the corresponding image points. With the help of those point pairs, one can establish a linear system of equations for which the constrained minimization problem needs to be solved. The most famous calibration algorithm based on this approach was established by Roger Tsai [Tsa86]. However, there are also nonlinear approaches to estimate the intrinsic parameters. A thorough mathematical comparison of both the linear and the nonlinear approach including numerical examples can be found in the book "Three-Dimensional Computer Vision" by Olivier Faugeras [Fau93, page 51-68].

In this project, the intrinsic parameters were estimated with the help of the free software "JCamCalib". This software was developed by Hugo Ortega Hernández for the purpose of his master's thesis "Visual Tracking of Multiple Objects with Mobil Camera in Dynamic Environments" [Her]. The program comes with a printable chessboard pattern with uniform dimensions. As input, the program receives a series of images of this chessboard pattern, where the pattern has different positions and orientations towards the camera in every image. In theses images, the cross points of the chessboard's squares are located, as it is shown in figure 19. Based on the acquired data, the software then numerically calculates the projection matrix P_{pro} (see equation 4.12). The following was the result of the calibration:

$$P_{pro} = \begin{pmatrix} 859.01647949 & 0 & 337.31808472 & 0 \\ 0 & 860.22406006 & 288.23727417 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.13}$$

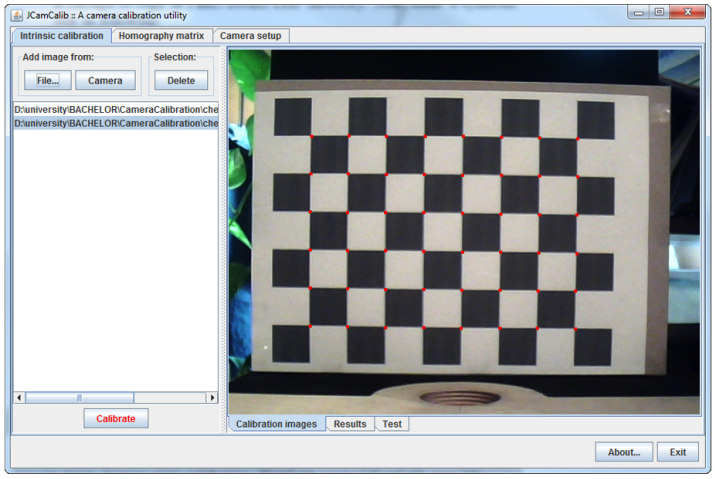

Figure 19: *The Chessboard Pattern with Detected Cross Points in JCamCalib*

4.2 Extrinsic Parameters

As previously mentioned, the intrinsic parameters provide a relation between image coordinates and camera coordinates. Nonetheless, it is not advisable to describe the coordinates of a certain point in space, for instance those of a marker's center point in the scene, in camera coordinates because these coordinates are dependent on the camera. It makes more sense to describe the position of a point within a world coordinate system that can be defined at will. The camera itself will also have a position and an orientation within this world coordinate system that can be defined by exactly six parameters. Those are the translation of the camera along the x-axis t_x, the translation along the y-axis t_y, the translation along the z-axis t_z, as well as the rotation angles around the x-axis r_x, around the y-axis r_y and around the z-axis r_z. These parameters are called extrinsic parameters. Consequently, the extrinsic parameters describe the relation between the world coordinate system and the camera coordinate system.

4.2.1 Defining the World Coordinate System

The world coordinate system is the coordinate system that is used to describe the markers' positions within the scene. Figure 20 (a) illustrates how the world coordinate system was defined for the stop-motion capture build-up. The puppet is standing in its 0° position. The origin of the world coordinate system marks the point where the puppet's turning axis meets the platform. The y-axis is equal to the puppet's turning axis. The x-axis, when seen from a frontal view, points to the right. Forming a right handed

coordinate system, the z-axis consequently points to the camera. The axes are scaled in millimeters.

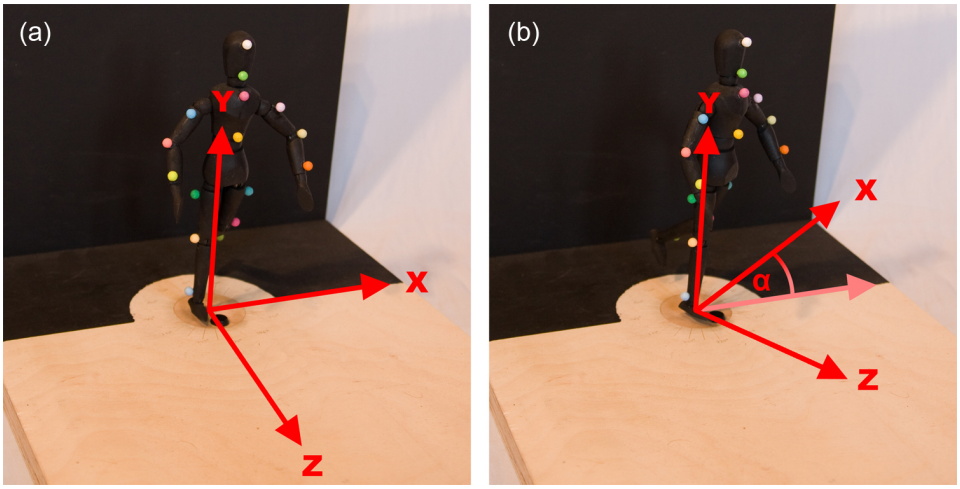

Figure 20: *(a) The World Coordinate System with the Puppet in its 0° Position, (b) The World Coordinate System in the 30° Position*

As described in section 2.1, the build-up is designed in a way that does not require the camera to be turned around the puppet, but for the puppet to be turned around its own axis. However, the coordinates that are supposed to be changed when the puppet is turned are not those of the puppet (respectively its markers), but those of the camera. To accomplish this, the world coordinate system needs to turn along with the puppet. Figure 20 (b) is an exemplary example that shows the world coordinate system with the puppet standing in its 30° position.

4.2.2 Estimating the Extrinsic Parameters

With the knowledge of both the world coordinate system and the distances between the camera and the puppet, as they are pointed out in section 2.1.3, one can now determine the extrinsic camera parameters for the system. The puppet can be turned by 360° at 30° intervals. This is equivalent to 12 cameras that are distributed along a circular path around the puppet at 30° intervals. For each of those cameras, the extrinsic parameters can be determined depending on the rotation angle α of the puppet, as described in the following:

The y-axis of the world coordinate system is always pointing up, while the y-axis of the camera is always pointing down (see figure 18). This can be described as a rotation of the camera around the x-axis of the world coordinate system by $r_x = 180°$. The camera's

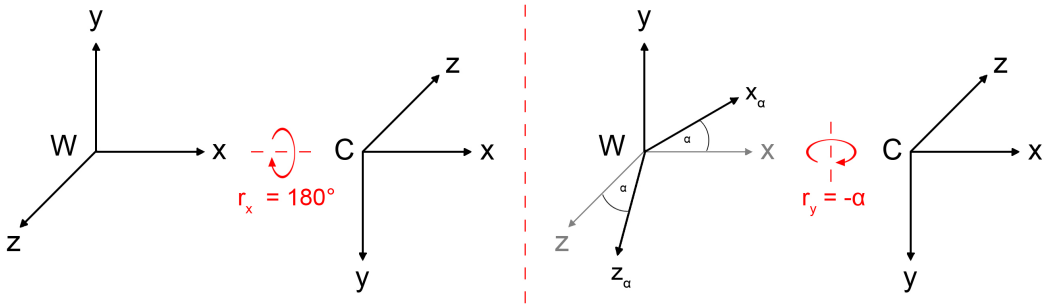

Figure 21: *Rotation Angles of the Camera*

rotation around the y-axis of the world coordinate system depends on the rotation angle α of the puppet. It is $r_y = -\alpha$. Because both camera coordinate system and world coordinate system are standing perpendicular on the bottom panel of the build-up, two rotations are sufficient to describe the camera's orientation. Therefore, it is $r_z = 0°$. Figure 21 points out the single rotation angles of the camera.

The y-translation of the camera ($t_y = 80mm$) does not depend on the rotation of the puppet and is directly given by the build-up's dimensions. As figure 22 illustrates the camera's translations along the x-axis and z-axis of the world coordinate system are:

$$t_x = -400\text{mm } sin(\alpha) \tag{4.14}$$

$$t_z = 400\text{mm } cos(\alpha) \tag{4.15}$$

The following table summarizes all extrinsic parameters (the translations rounded to millimeters) for the 12 camera positions (respectively all 12 turning angles α for the puppet):

α	r_x	r_y	r_z	t_x	t_y	t_z	α	r_x	r_y	r_z	t_x	t_y	t_z
[degrees]				[mm]			[degrees]				[mm]		
0	180	0	0	0	80	400	180	180	180	0	0	80	-400
30	180	-30	0	-200	80	346	210	180	150	0	200	80	-346
60	180	-60	0	-346	80	200	240	180	120	0	346	80	-200
90	180	-90	0	-400	80	0	270	180	90	0	400	80	0
120	180	-120	0	-346	80	-200	300	180	60	0	346	80	200
150	180	-150	0	-200	80	-346	330	180	30	0	200	80	346

With those parameters one can now easily express the transformation from camera

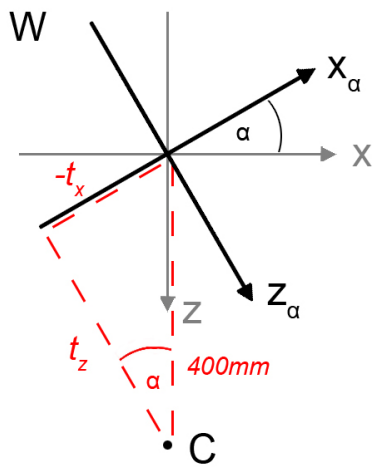

Figure 22: *Translations of the Camera in an Overhead View*

coordinates to world coordinates as the following matrix:

$$M_W^C(\alpha) = T \begin{pmatrix} -400\text{mm } sin(\alpha) \\ 80mm \\ 400\text{mm } cos(\alpha) \end{pmatrix} R_y(-\alpha) R_x(180°) \tag{4.16}$$

Consequently, the transformation from world coordinates to camera coordinates is:

$$\begin{aligned} M_C^W(\alpha) &= M_W^C(\alpha)^{-1} \\ &= R_x(180°) R_y(\alpha) T \begin{pmatrix} 400\text{mm } sin(\alpha) \\ -80mm \\ -400\text{mm } cos(\alpha) \end{pmatrix} \end{aligned} \tag{4.17}$$

Finally, one can now predict onto what point p^B within the image a given point p^W within the world coordinate system is projected:

$$p^B = P_{pro} M_C^W(\alpha) p^W \tag{4.18}$$

5 Reconstructing the Marker Positions

The stop-motion capture system is, as pointed out in section 3, capable of detecting the position of a marker within a picture taken by the camera. With the knowledge of the calibration parameters of the camera, as they were acquired in section 4, it is now possible to reconstruct the position of this marker in space or, more precisely, in world coordinates. This requires the marker to be detected in at least two different images (taken from two different camera positions). The concept used to accomplish this is shown in figure 23. With the help of the calibration parameters one can construct the back projection line of the marker on each camera image. The actual position of the marker must lie somewhere on the back projection line. With two back projection lines of the same marker, its position must consequently be the point of intersection of both lines.

The following points out this approach in detail, assuming one does not have just two, but N cameras. Thereby, the first section describes the coordinate transformations that are required. The second section shows how to construct the back projection lines while the last section points out how to practically approximate the marker position with the help of the back projection lines.

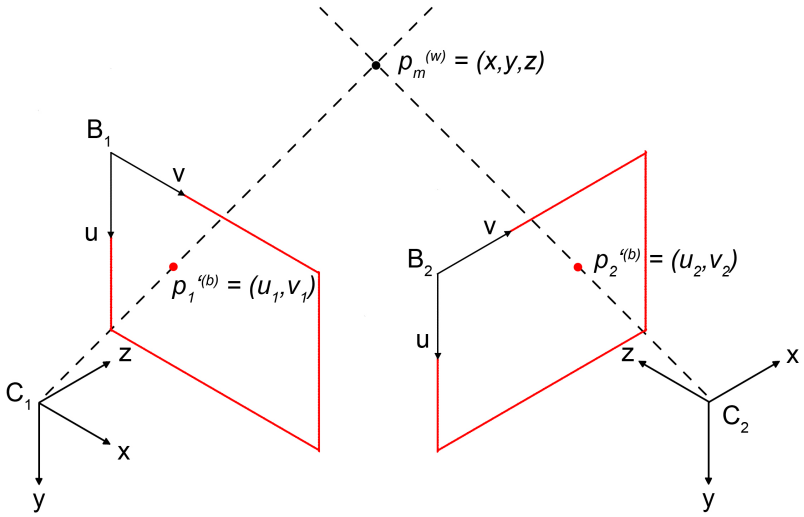

Figure 23: *The Back Projection Lines Intersect in the Position of the Marker*

5.1 Coodinate Transformations

Due to the fact that the marker position needs to be defined within the world coordinate system, the back projection lines must also be defined in world coordinates. First of all, it is necessary to transform the projection points $p_i^{\prime(b)}$ of each camera image i into world coordinates.

5.1.1 Image to Camera

The first step is a transformation from image coordinates to camera coordinates, as they were defined in section 4.1.2. Equation 4.10 and 4.11 describe the conversion from the x- and y-coordinates of the camera coordinate system to the u- and v-coordinates of the image. Rearranged to x and y, one has:

$$x = \frac{u - u_0}{c_{scale}} z \tag{5.1}$$

$$y = \frac{v - v_0}{c_{scale}} z \tag{5.2}$$

In both equations, z is at first unknown. According to the pinhole model (see figure 18), the projective plane of the camera is at $z = f$. Because p_i' lies within the projective plane, the coordinates of $p_i^{\prime(c)}$ must be:

$$x = \frac{u - u_0}{c_{scale}} f \tag{5.3}$$

$$y = \frac{v - v_0}{c_{scale}} f \tag{5.4}$$

$$z = f \tag{5.5}$$

The focal length f of the camera is specified by its manufacturer as $f = 4.22mm$. Notice here that the actual effective focal length of the camera does not result from the calibration. Only the product $c_{scale} = fk$ (see equation 4.13) is known. However, as it will be pointed out in section 5.2, the precise knowledge of f (respectively z) in equations 5.3 and 5.4 is not relevant to construct the back projection line.

5.1.2 Camera to World

The next step is the transformation from camera coordinates into world coordinates. The required transformation matrix was already estimated in equation 4.16. It is therefore:

$$p_i'^{(w)} = M_W^C p_i'^{(c)} \tag{5.6}$$

5.2 Constructing the Back Projection Lines

The positions of the projection points $p_i'^{(w)}$ of the marker are now known in world coordinates. From the calibration in section 4, the camera positions $c_i^{(w)}$ are also known. They are:

$$c_i^{(w)} = \begin{pmatrix} -400\text{mm } sin(\alpha_i) \\ 80mm \\ 400\text{mm } cos(\alpha_i) \end{pmatrix} \tag{5.7}$$

As shown in figure 24, the back projection lines for each projection point can now be described as:

$$\begin{aligned} S_i(t) &= c_i^{(w)} + (p_i'^{(w)} - c_i^{(w)})t \\ &= c_i^{(w)} + \mathbf{v_i}t \end{aligned} \tag{5.8}$$

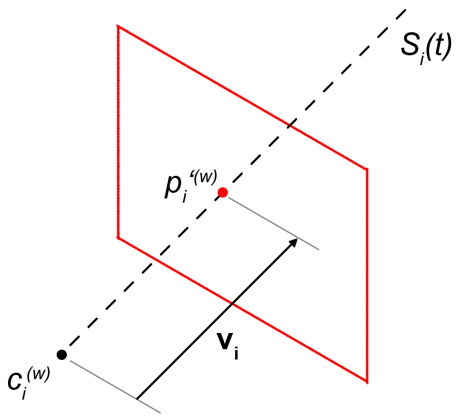

Figure 24: *Constructing a Back Projection Line*

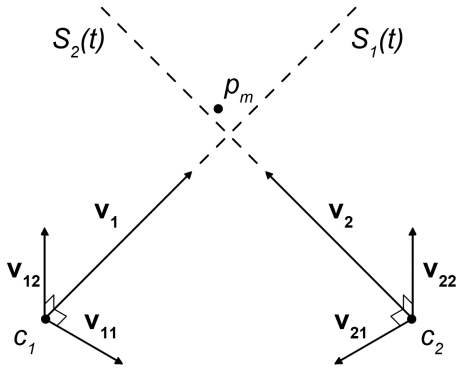

Figure 25: *Estimating the Marker Position*

The back projection line $S_i(t)$ now contains all possible points in space that are projected onto point p_i' by the camera C_i. This also explains why it is not necessary to

know f precisely in equations 5.3 and 5.4. Due to the intercept theorems, every point $p_z = (x_z, y_z, z_z)$ that is returned by equations 5.3 to 5.5, lies exactly on the back projection line and is therefore suitable to construct $S_i(t)$ according to equation 5.8.

Having $S_i(t)$ for a least two cameras C_i, it is now possible to construct a linear system of equations to return t_m. The point of intersection $S(t_m)$ would then be equal to the position of the marker $p_m^{(w)}$. However, due to measuring errors, the back projection lines are unlikely to intersect at one point. Therefore, a more useful approach is to find the point p_{apr}, that has the minimum distance to each available back projection line, thus approximating p_m as good as possible. The following section explains this approach.

5.3 Estimating the Marker Position

The next task is to find a point p_{apr} that has the minimum distance from a series of lines $Si(t)$. The approach to do so is founded on the following idea:

A line in space is given as the intersection of two planes. To construct these planes for a given line $S_i(t)$, one can do the following:

$$\mathbf{v_{i1}} = \frac{\mathbf{v_i} \times \mathbf{v_{iL}}}{\|\mathbf{v_i} \times \mathbf{v_{iL}}\|} \tag{5.9}$$

$$\mathbf{v_{i2}} = \frac{\mathbf{v_{i1}} \times \mathbf{v_i}}{\|\mathbf{v_{i1}} \times \mathbf{v_i}\|} \tag{5.10}$$

In equation 5.9 and 5.10, $\mathbf{v_i}$ and $\mathbf{v_{i1}}$, as well as $\mathbf{v_i}$ and $\mathbf{v_{i2}}$ define two planes that are perpendicular to one another. Notice that $\mathbf{v_{iL}}$ can be any vector that is linearly independent from $\mathbf{v_i}$. The intersection of those planes is exactly $S_i(t)$. Figure 25 illustrates this. With the dot products $\langle \mathbf{v_{i1}}, \mathbf{p_{apr}} \rangle$ and $\langle \mathbf{v_{i2}}, \mathbf{p_{apr}} \rangle$, one can now make a statement about the distance of p_{apr} to the center of the coordinate system that is defined by $\mathbf{v_i}$, $\mathbf{v_{i1}}$ and $\mathbf{v_{i2}}$. For the center to be at c_i, the minimum distance must be given by $\langle \mathbf{v_{i1}}, \mathbf{c_i} \rangle$ and $\langle \mathbf{v_{i2}}, \mathbf{c_i} \rangle$. This can be described with the following linear system of equations:

$$A\mathbf{x} = \mathbf{y}$$

$$\begin{pmatrix} v_{11x} & v_{11y} & v_{11z} \\ v_{12x} & v_{12y} & v_{12z} \\ \vdots & \vdots & \vdots \\ v_{n1x} & v_{n1y} & v_{n1z} \\ v_{n2x} & v_{n2y} & v_{n2z} \end{pmatrix} \begin{pmatrix} x_x \\ x_y \\ x_z \end{pmatrix} = \begin{pmatrix} \langle \mathbf{v_{11}}, \mathbf{c_1} \rangle \\ \langle \mathbf{v_{12}}, \mathbf{c_1} \rangle \\ \vdots \\ \langle \mathbf{v_{n1}}, \mathbf{c_n} \rangle \\ \langle \mathbf{v_{n2}}, \mathbf{c_n} \rangle \end{pmatrix} \tag{5.11}$$

In 5.11, $\mathbf{x}$ is equivalent to p_{apr}. Solving this equation for $\mathbf{x}$ requires finding the pseudo inverse of A. To do so, one can left multiply each side by A^T, then by $(A^T A)^{-1}$:

$$(A^T A)^{-1} A^T A\mathbf{x} = (A^T A)^{-1} A^T \mathbf{y} \tag{5.12}$$

The first part of the left hand side of 5.12 is now a matrix multiplied by its inverse which by definition is the identity matrix I. Because $IM = M$ for all matrices M, one can write:

$$\mathbf{x} = (A^T A)^{-1} A^T \mathbf{y} \tag{5.13}$$

The exact implementation of the algorithm explained in this section can be found in the *Reconstruct3D.java* class (See section 7.2 for more details on the software architecture). For the required linear algebra, the Java library "JAMA" [Mat] was used.

6 Reconstructing the Puppet Pose

The positions of the markers on the puppet are now known. The next step is to use this information to reconstruct the puppet's pose. Therefore, one first needs a model that is suitable to describe that pose sufficiently. In computer animation, or more precisely character animation, it is customary to model humans, animals and the like as hierarchic structures that can be interpreted as a sort of skeleton. Once an appropriate skeleton for the puppet is defined, any pose can be mapped onto that skeleton. Thus, the task of reconstructing the pose is equal to the task of (re-)constructing a suitable skeleton.

The first part of this section introduces the idea of modeling skeletons as hierarchic structures while the second part describes how the skeleton for this system's puppet was defined. The last part then explains how to implement the skeleton as a COLLADA file that can be interpreted by the JMonkey Engine.

6.1 Skeletons as Hierarchic Structures

A common approach to quickly describe a man's pose in a simple manner is to draw a stick-figure. If one imagines the stick-figure's lines to be a series of linked bones, one has basically already created the essence of the model that is actually used in computer animation to model humans or humanoid creatures for animation purposes. These creatures or characters are represented by a simplified skeleton, where the skeleton's "bones" (not in a biological sense) are organized in a hierarchic structure. Figure 26 illustrates such a structure. Every node within the tree structure on the right side contains information that is assigned to the corresponding bone of the skeleton on the left side. Node A for instance could contain information about the length of lower leg's bone, as well as information about its orientation, for instance represented by a directional vector. When this directional vector is changed, the skeleton moves its lower leg. The hierarchy of the nodes thereby creates a kinematic chain. The foot can move relative to the lower leg that moves relative to the upper leg and so on. The top of the hierarchy is defined as the root node. The root node plays a special role and can, for instance, not be directly interpreted as a bone. Nonetheless, the root node can contain information about the skeleton's position within a reference coordinate system and can therefore be used to translate or move the whole skeleton along that system.

Even though the stop-motion capture system is not directly interested in animation, it makes sense to make use of such a "skeletal system" to describe the puppet's pose.

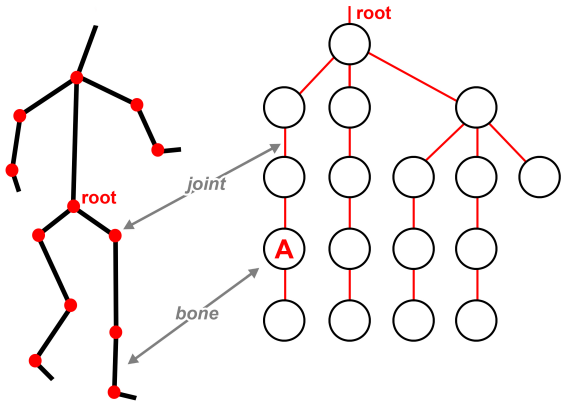

Figure 26: *A Skeleton Represented as a Hierarchic Structure*

6.2 Defining the Skeleton

The question now is how to describe the puppet used in this project as an appropriate skeletal system.

Due the fact that the puppet is equipped with jointed upper arms, lower arms and hands, as well as upper legs, lower legs and feet, it makes sense to model those limbs as corresponding bones. In order to do justice to the puppet's physique, additional bones in the shoulder and pelvis area were necessary.

Figure 27 (a) shows the skeleton including the names for all bones. The illustration of the bones is thereby adopted by the skeletal system of the JMonkey Engine (see figure 27 (b)) and differs from the schematic skeleton from figure 26. In JMonkey, a bone is represented as a line that always has a sphere at its end to represent the joint. This is the reason why, for example, the head bone also ends with a joint even though there are no further bones attached to it. The naming of the bones conventionally goes by the attached joints, which can be misleading. The *l-elbow-joint* for instance describes the entire upper arm bone, excluding the shoulder joint and including the elbow joint.

With figure 27 (a), the hierarchy of the bones is already defined. What must be determined next are the lengths of the bones. Those were approximated by measuring the puppet's limbs. It is less important to determine the exact numerical values than to obtain the ratio of the bones because the absolute size of the puppet is irrelevant to describe the pose.

Bone hierarchy and lengths are independent from the puppet's pose and were defined once for the entire system as it was described above. As previously mentioned, the puppet's pelvis is fixed. Therefore, the *lower-back-joint* bone, the *r-hip-joint* bone and

the *l-hip-joint* bone do not have any degrees of freedom and were also defined once for the entire system.

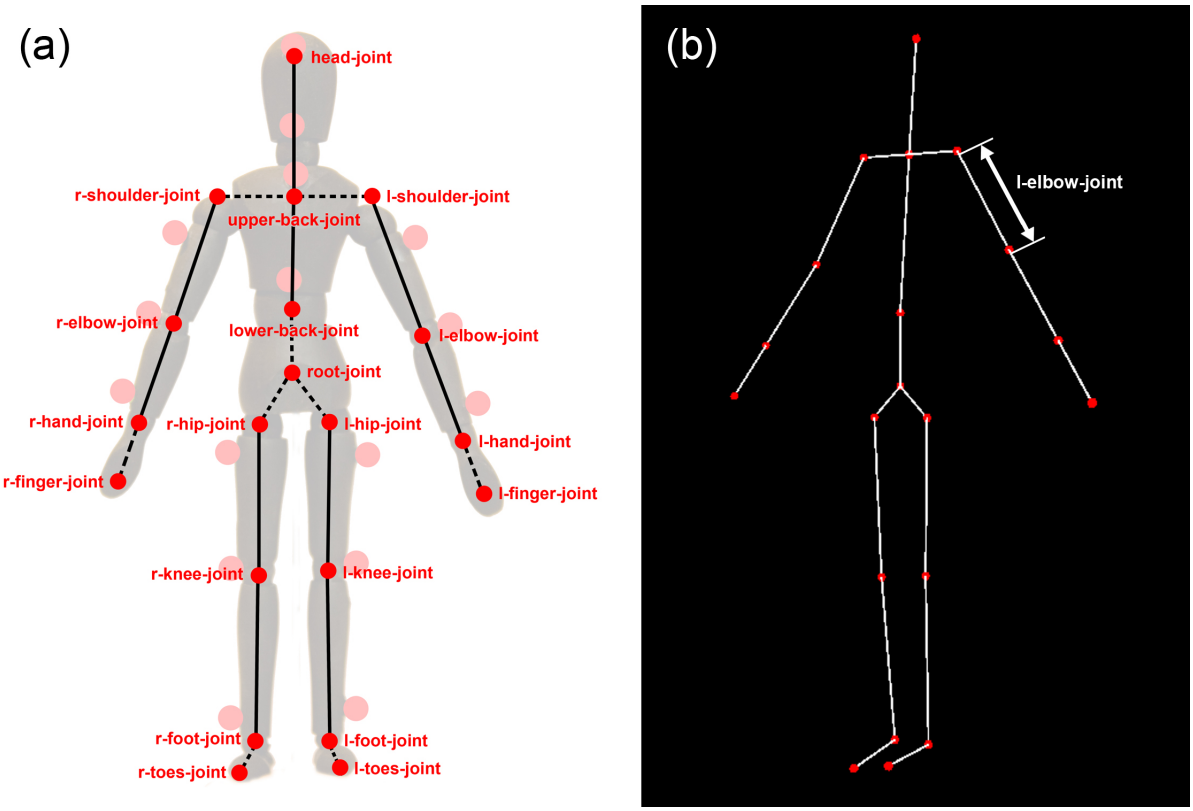

Figure 27: *(a) The Skeleton of the Puppet, (b) The Skeleton Represented in the jMonkeyEngine Skeletal System*

The other bones' orientations now depend on the pose. How those orientations are described and estimated depends on the file format that is used to describe the skeleton. For this project the COLLADA format was used. Section 6.3 describes how to build the skeleton as such a COLLADA file.

6.3 Building the COLLADA File

COLLADA [Gro], short for COLLAborative Design Activity, is an open standard interchange file format for 3D applications. It is managed by the non-profit technology consortium, the Khronos Group. The file format defines an open standard XML schema [Gro08] to describe various data for graphics software applications.

There were several reasons to use the COLLADA format to rebuild the skeleton of the stop-motion capture puppet:

- The JMonkeyEngine provides an importer for COLLADA files
- Most industry standard 3D applications support COLLADA which provides the opportunity to import the captured pose/skeleton into any of those programs for further usage
- The whole pose/skeleton can be represented in a few lines of XML that can also be interpreted by humans

The design of the COLLADA file is very simple. The hierarchic structure of the skeleton is achieved by nesting the XML elements accordingly. Those elements are called *node* elements. Based on the hierarchy defined in section 6.2, the XML markup has the following form:

```
<node id="root-joint" type="JOINT">
   <node id="l-hip-joint" type="JOINT">
      ...
      <node id="l-toes-joint" type="JOINT"/>
   </node>
   <node id="r-hip-joint" type="JOINT">
      ...
      <node id="r-toes-joint" type="JOINT"/>
   </node>
   <node id="lower-back-joint" type="JOINT">
      ...
   </node>
</node>
```

Every *node* element with the exception of the *root-joint* contains a *matrix* element. The *matrix* element's content is a homogeneous 4×4 matrix. This matrix describes the translation of the current bone, or more precisely of the joint at the end of that bone. For instance, the following excerpt describes that the *l-elbow-joint* is translated relatively to the *l-shoulder-joint* by $t_x = 3.2703316$, $t_y = -0.31502795$ and $t_z = -0.3093352$:

```
<node id="l-shoulder-joint" type="JOINT">
   <matrix>...</matrix>
   <node id="l-elbow-joint" type="JOINT">
      <matrix>1 0 0 3.2703316 0 1 0 -0.31502795 0 0 1 -0.3093352 0 0 0 1</matrix>
   </node>
</node>
```

Therefore, in order to build the skeleton with the desired pose, one needs to estimate the translation matrix m_{trans} for every bone. This is a rather simple step because in

COLLADA those translation matrices do not refer to different local coordinate systems, but to the same world coordinate system. Due to the fact that the marker positions were also defined in reference to one world coordinate system, the directional vector $\mathbf{a_{dir}}$ for each bone can be estimated as described in equation 3.1 and 3.2 of section 3.1.3:

$$\mathbf{a_{dir}} = \frac{m_2 - m_1}{\|m_2 - m_1\|} \tag{6.1}$$

The translation vector for each bone depending on the bone length l_{bone} is therefore:

$$\mathbf{a_{trans}} = l_{bone}\mathbf{a_{dir}} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{6.2}$$

This conforms to the following translation matrix M_{trans}:

$$M_{trans} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.3}$$

The following table provides an overview of the assignments between the different markers and bones, as they were used for equation 6.1:

limb	bone	m_1	m_2
head	head-joint	head.2	head.1
torso	upper-back-joint	torso.2	torso.1
upper arm, left	l-elbow-joint	arm.L1	arm.L2
lower arm, left	l-hand-joint	arm.L2	arm.L3
upper arm, right	r-elbow-joint	arm.R1	arm.R2
lower arm, right	r-hand-joint	arm.R2	arm.R3
upper leg, left	l-knee-joint	leg.L1	leg.L2
lower leg, left	l-foot-joint	leg.L2	leg.L3
upper leg, right	r-knee-joint	leg.R1	leg.R2
lower leg, right	r-foot-joint	leg.R2	leg.R3

To complete the skeleton, one needs to find useful directional vectors for those bones that have no assigned markers. For this system, each hand (*l-finger-joint* and *r-finger-*

joint) was given the same directional vector as the lower arm (*l-hand-joint* and *r-hand-joint*) it is attached to. In order for the feet (*l-toes-joint* and *r-toes-joint*) to point forwards and be perpendicular to the lower legs (*l-foot-joint* and *r-foot-joint*), their directional vectors were estimated as the cross product of the lower legs' directional vectors and a vector that is parallel to the x-axis:

$$\mathbf{a}_{\mathbf{dir}}^{\mathbf{foot}} = \frac{\mathbf{a}_{\mathbf{dir}}^{\mathbf{leg}} \times \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T}{\left\| \mathbf{a}_{\mathbf{dir}}^{\mathbf{leg}} \times \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T \right\|} \tag{6.4}$$

The same was done for the shoulder bones (*l-shoulder-joint* and *r-shoulder-joint*) so that they are perpendicular on the *upper-back-joint* and point towards the left, respectively the right:

$$\mathbf{a}_{\mathbf{dir}}^{\mathbf{l-shoulder}} = \frac{\mathbf{a}_{\mathbf{dir}}^{\mathbf{upper-back}} \times \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T}{\left\| \mathbf{a}_{\mathbf{dir}}^{\mathbf{upper-back}} \times \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T \right\|} \tag{6.5}$$

$$\mathbf{a}_{\mathbf{trans}}^{\mathbf{r-shoulder}} = \frac{\mathbf{a}_{\mathbf{dir}}^{\mathbf{upper-back}} \times \begin{pmatrix} 0 & 0 & -1 \end{pmatrix}^T}{\left\| \mathbf{a}_{\mathbf{dir}}^{\mathbf{upper-back}} \times \begin{pmatrix} 0 & 0 & -1 \end{pmatrix}^T \right\|} \tag{6.6}$$

Notice that these are simplifications that only make sense as long as the torso or the legs do not twist. This issue is dealt with in section 8.2.

7 About the Software

This section introduces the application software "JumpingJack 3D Capturer" that was developed for this project. The software offers a single graphical user interface for operating the entire stop-motion capture system and implements all calculations and algorithms that were pointed out in the previous chapters.

The first part of this section describes the basic features that the software offers to the operator while the second part provides a short insight of the software architecture with the help of a simplified class diagram.

7.1 Software Features

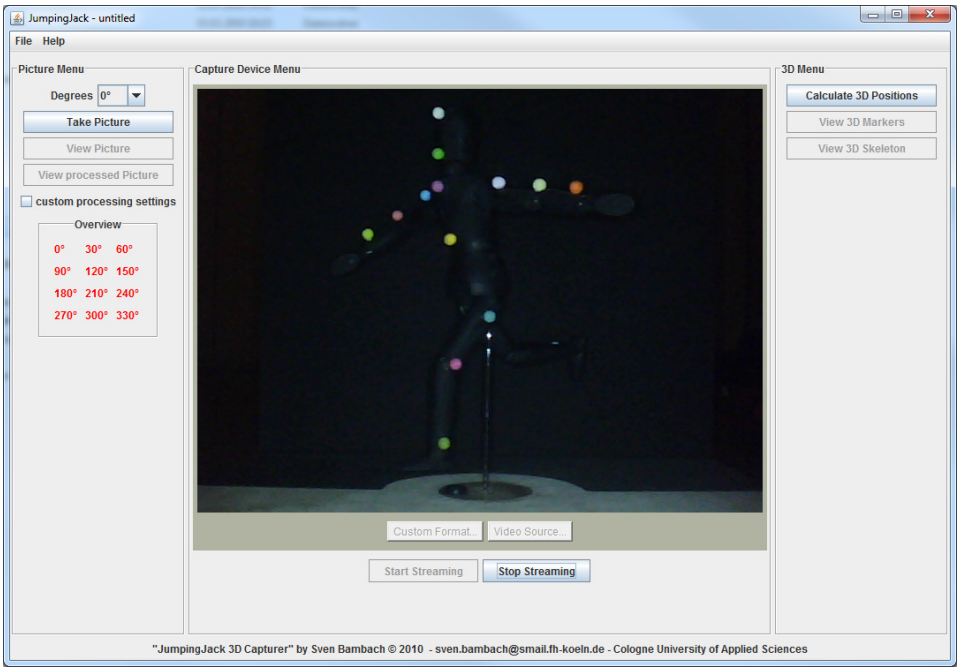

Figure 28: *The Interface of the "JumpingJack 3D Capturer" Software*

The "JumpingJack 3D Capturer" software provides all the functions that are needed to operate the stop-motion capture system in one graphical user interface. The features of this interface are pointed out in section 7.1.1 while describing the general usage of the software. Section 7.1.2 explains the "Jumping Jack" file format that was created specifically for this software. It enables the operator to easily save all images, as well as the data that is required to recalculate the captured pose, as one project. This project can later be loaded into the program and worked with without the need of the build-

up's webcam being installed. Lastly, section 7.1.3 explains how the captured pose can be exported as a COLLADA file that can be opened in almost any industry standard 3D software application.

7.1.1 General Usage

Figure 28 shows the user interface after launching the program and starting the webcam stream. The interface is divided into three menus. The "Picture Menu" on the left enables the operator to take a picture of the current webcam stream. The "Capture Device Menu" displays the webcam stream and also allows certain webcam adjustments (e.g. changing the streaming format or picking the right webcam as a source) if needed. The "3D Menu" on the right will later show the reconstructed positions of each marker and enable the operator to open different 3D views of the puppet.

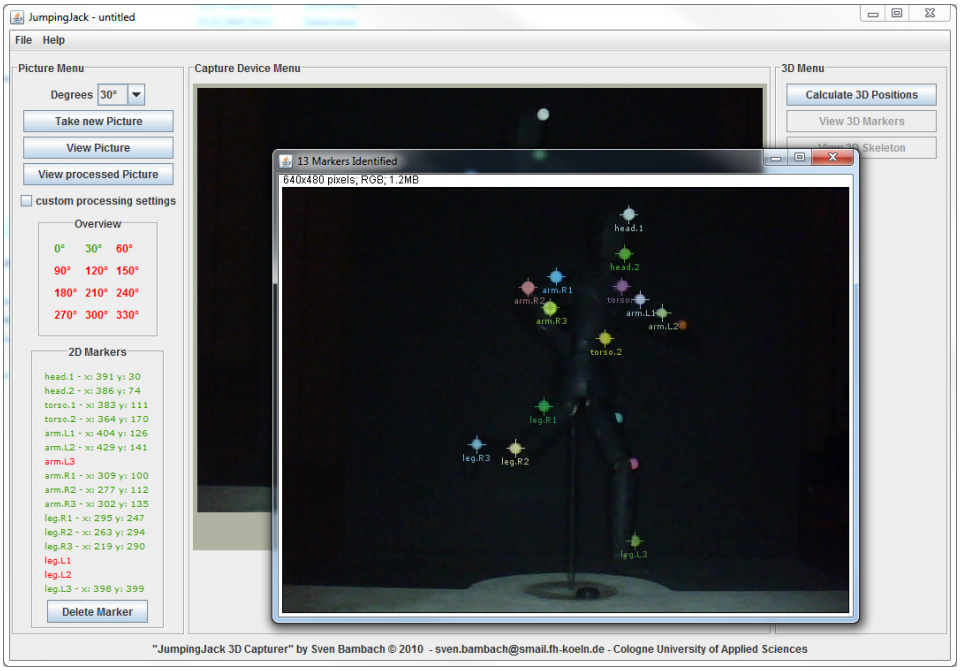

Figure 29: *A Pop-Up Window Displays the Processed 30° Picture*

Figure 29 shows the interface right after the picture for the 30° position was taken. The picture is immediately processed and a pop up window displays the processed picture with all identified markers for the operator to check. The "2D Markers" panel within the "Picture Menu" additionally lists the markers of the current picture, as well as their positions. The panel also provides a button to delete single markers in the image in case they have been falsely identified. The "Overview" panel highlights all pictures that have

already been taken.

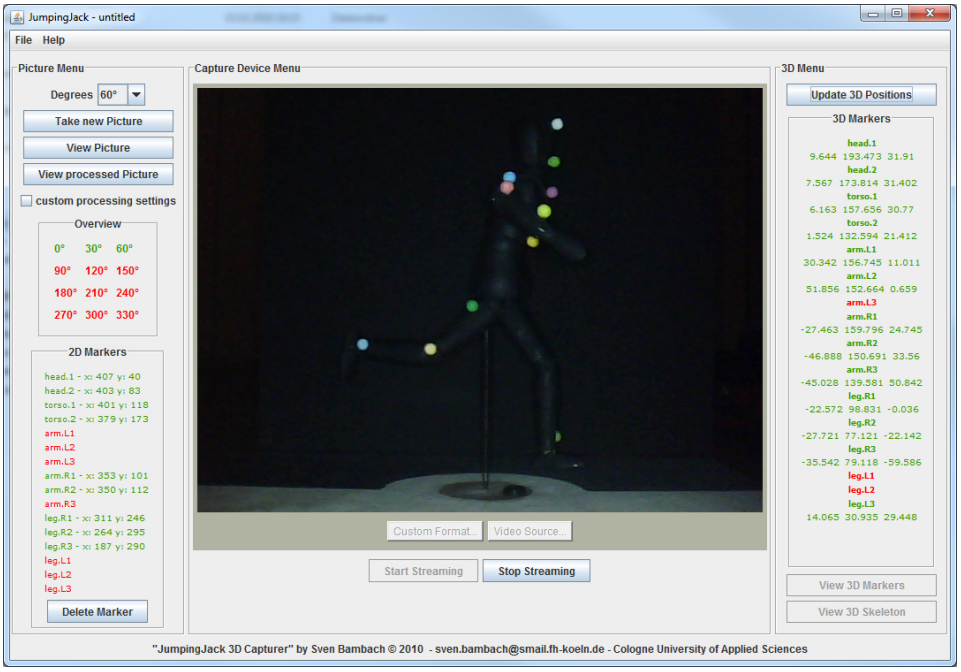

Figure 30: *The "3D Markers" Panel on the Right Lists Up the Markers' Positions*

When pressing the "Calculate 3D Positions" button in the "3D Menu", a list of all the markers and their estimated 3D positions will appear. This can be seen in figure 30. The positions can be updated at any time and will be recalculated based on the amount of the current 2D data. In figure 30, for instance, the markers "arm.L3", "leg.L1" and "leg.L2" have not been identified on at least two different images yet and are therefore marked red.

Once all marker positions are calculated, the operator can launch two different 3D views via the buttons in the "3D Menu". The "3D Markers" view is shown in figure 31 (a). It shows all the markers in space as well as, for orientation, the turning axis of the puppet. This can be view from 360° by using the left and right arrow keys on the keyboard. This lets the operator easily check and validate the data because a falsely identified marker in one of the pictures would cause the corresponding 3D marker to be noticeably out of line in this view.

The "3D Skeleton" view is shown in figure 31 (b). It shows the skeleton, as it was defined in section 6.2, as it reconstructs the pose of the puppet. This can be viewed from 360° in the same way as the "3D Markers" view.

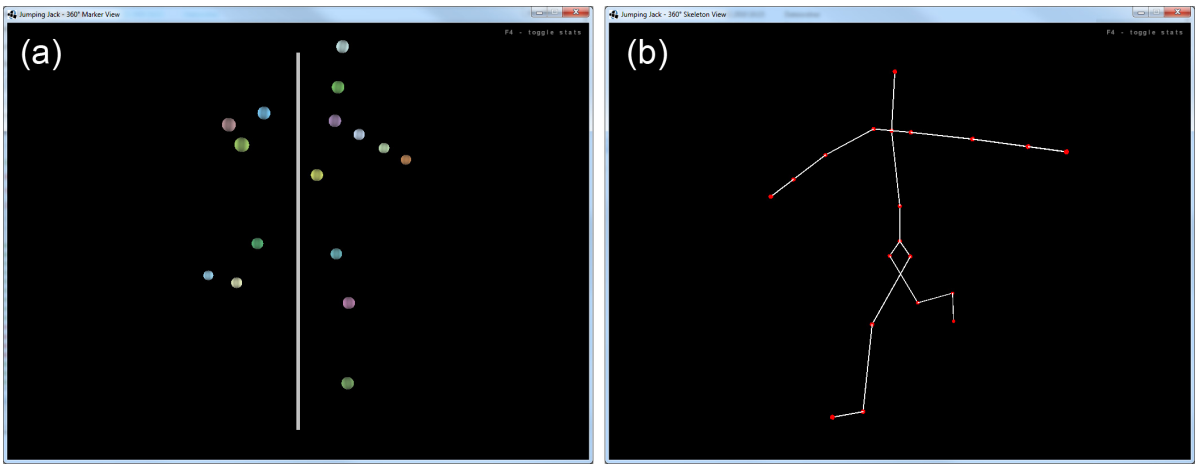

Figure 31: *(a) The 360° Marker View, (b) The 360° Skeleton View*

7.1.2 Saving and Loading Projects

The "JumpingJack 3D Capturer" software also offers the possibility to save or load projects via the menu bar. A project includes all images that were taken from the puppet, as well as the processed images and all identified markers and their locations within those images. The projects are saved in the "JumpingJack" file format that defines an XML markup specifically designed for this software. The file has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<JJProject name="my_project.juja">
   <image name="0" normal_src="my_project_0.jpg" pro_src="my_project_0p.jpg">
      <marker name="leg.L1" x="380" y="267" />
      <marker name="head.1" x="349" y="23" />
      ...
   </image>
   <image name="30" normal_src="my_project_30.jpg" pro_src="my_project_30p.jpg">
      <marker name="leg.R1" x="313" y="262" />
      ...
   </image>
   ...
</JJProject>
```

The images that belong to the project are automatically saved as JPEG files in the same directory as the "JumpingJack" file. The 3D data is not part of the project, as it will be calculated while the program is running. Nevertheless, the "JumpingJack 3D Capturer" software provides the option of exporting the pose of the puppet, as it will be pointed out in section 7.1.3.

7.1.3 Exporting the Puppet Pose

As explained in section 6.3, the skeleton that represents the puppet's pose is written in the COLLADA file format that can then be interpreted by the JMonkeyEngine. Via the menu bar, the software also offers the possibility to export the skeleton and save it as a COLLADA file that can then be imported into almost any industry standard 3D software application for further usage. The structure of such a file is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.0">
    <asset>
        <contributor>
            <authoring_tool>JumpingJack Collada Builder - feedback to sven.
                bambach@smail.fh-koeln.de</authoring_tool>
        </contributor>
        <created>Fri Oct 15 17:37:04 CEST 2010</created>
        <unit meter="1.0" name="centimeters" />
        <up_axis>Y_UP</up_axis>
    </asset>
    <library_visual_scenes>
        <visual_scene id="skeleton" name="skeleton">
            <node id="root-joint" type="JOINT">
                ...
            </node>
        </visual_scene>
    </library_visual_scenes>
    <scene>
        <instace_visual_scene url="#skeleton" />
    </scene>
</COLLADA>
```

The hierarchy of the *node* elements has already been explained in section 6.3.

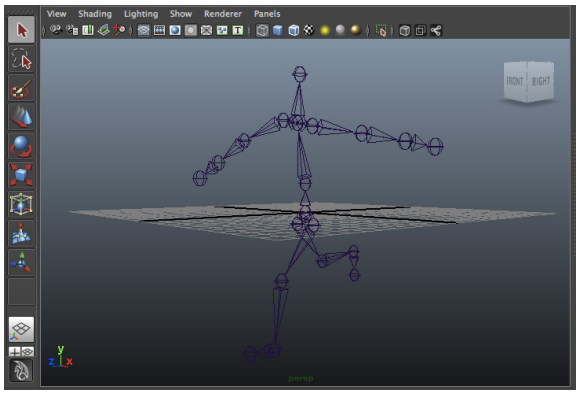

Figure 32: *A JumpingJack COLLADA Skeleton Imported Into Maya 2011*

Figure 32 shows the skeleton from Figure 31 (b) after being imported into "Autodesk Maya 2011".

7.2 Software Architecture

Figure 33 shows a class diagram of the software. This diagram was exclusively designed for the purpose of pointing out the basic architecture of the software in association with the following explanations in this section. As a result of this, the diagram was simplified in some ways. It does not show class attributes or visibilities and it only shows those methods that are mentioned in the explanation. Some classes are left out completely to keep the diagram concise. If not labeled otherwise, the association arrows imply that a class holds the given number of instances of the class it's pointing to as class attributes.

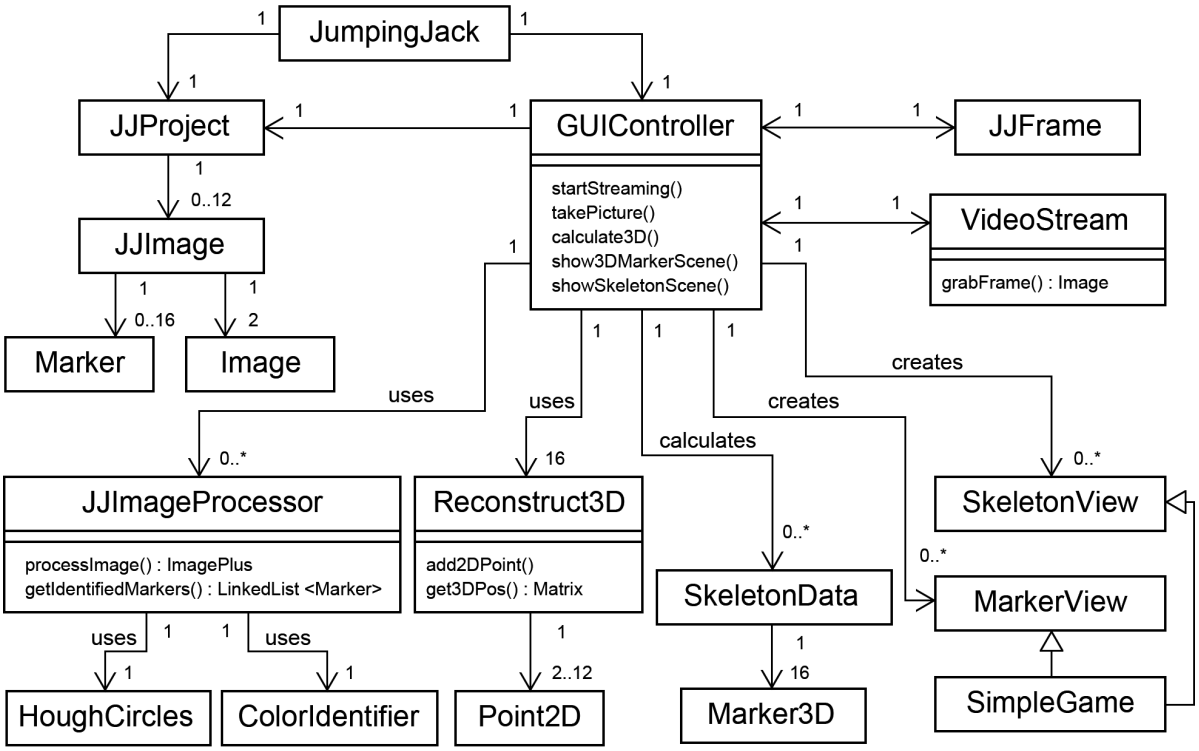

Figure 33: *The Simplified Class Diagram of the "JumpingJack 3D Capturer" Software*

The following explains the basic interaction of the components in the diagram that occurs while operating the software:

- The *JumpingJack* class is the main class that initiates the program. It holds an instance of the *JJProject* class that stores the data of the current project, as

well as an instance of the *GUIController* class. Once instantiated, the *GUIController* class immediately opens the user interface window (*JJFrame*) and waits for user interaction. Every listener in the *JJFrame* class calls a method within the *GUIController* class.

- The *startStreaming()* method initiates the webcam stream. When the operator takes a picture with the "Take Picture" button, the *GUIController*'s *takePicture()* method first calls the *grabFrame()* method of its *VideoStream* instance which returns the current video stream frame as an *Image* object. The *takePicture()* method then instantiates a *JJImageProcessor* object that returns the processed image (*processImage()*) as well as a list of the markers that were identified in the taken picture (*getIdentifiedMarkers()*).

- This data is stored in the instance of the *JJProject* class. This instance holds one *JJImage* instance for every picture taken. Every *JJImage* instance thereby stores both original and processed images, as well as a list of the markers' names and locations in these images, represented as *Marker* objects.

- When the operator presses the "Calculate 3D Positions" button, the *GUIController*'s *calculate3D()* method instantiates one *Reconstruct3D* object for every marker that was located in at least two images. Every available position and the corresponding camera position for the marker is added to the *Reconstruct3D* instance with the *add2DPoint()* method and stored in a list of *Point2D* objects. Then, the *get3DPos()* method calculates the 3D position of the marker and returns it to the *GUIController*. The *GUIController* then creates an instance of the *SkeletonData* object that holds a list of all the 3D markers as *Marker3D* objects, as well as all other required 3D data.

- Finally, when pressing the "View 3D Markers" or "View 3D Skeleton" button, the *GUIController*'s *show3DMarkerScene()* method or *show3DSkeletonScene()* method instantiates a *MarkerView* object or a *SkeletonView* object with the 360° view window described in figure 31 in section 7.1.1. Both views are subclasses of the JMonkeyEngine's *SimpleGame* class that implements all methods that are needed to render a 3D scene.

8 Conclusion

This section ends the thesis with a short summary of what was achieved, as well as a list of aspects that could be subjects of further improvement or research.

8.1 Accomplishments

First of all, it must be said that the stop-motion capture system worked and the goals that were introduced in section 1.2 were accomplished.

The puppet's markers were located without difficulty and the color identifications worked reliably. The markers were rarely identified incorrectly. In most cases these were noised-induced errors that could be solved by simply taking the same picture again. If needed, the system's software let the operator quickly visualize how each marker was identified and offered to delete wrong identifications so that the system's workflow was not substantially interrupted. Depending on the complexity of the pose, it took an average of three to five pictures from different angles to identify all 16 markers often enough to estimate their positions in space.

Due to the camera calibration, the marker positions could be recalculated with sufficient accuracy. Samples showed a maximum deviation between the actual position and its estimated value of about $\pm 2mm$ per coordinate. However, this did not have noticeable influences on the reconstructed pose.

The reconstruction of the pose, i.e. the skeleton, worked as well. The skeleton data could even be exported, as shown in section 7.1.3. There were some constraints on the choice of possible poses, which are pointed out in the next section.

8.2 Topics of Further Improvement

The system is experimental and therefore offers room for further improvement and research. The most important aspects are summarized below:

- *Marker placement*: The system is currently not able to register if some body parts are twisted. This is most noticeable with the puppet's torso. If the puppet's upper body is, for instance, turned (instead of tilted) to one side, this has no influence on the directional vector that is estimated based on the torso's markers. A possible approach to solve this would be the attachment of further markers.

- *Marker size*: Further improvement could be accomplished by using smaller markers that could be easily attached to the puppet's hands and feet as well. One research

approach would be to estimate how small the markers can become and still be projected onto enough pixels to recognize their structure as circular.

- *Color calibration and Color identification*: The color identification can certainly be improved to a great extend with a more systematic approach when choosing the colors. Different shades could, for instance, be specifically mixed with the goal of reaching the largest possible color difference.

- *Camera calibration and 3D reconstruction*: The camera calibration disregards the fact that the system's webcam's lens is not infinitely small as it was assumed in the pinhole model. The actual camera image therefore contains radial distortions that are not considered when locating the markers within the image. This can cause some inaccuracy in the 3D reconstruction which can be avoided by estimating appropriate coefficients that compensate the lens aberration.

  A statistical evaluation of system's accuracy when reconstructing the marker positions would be of interest as well.

References

[Bur06] Wilhelm Burger. *Digitale Bildverarbeitung*. Springer, 2006.

[Eid04] Horst M. Eidenberger. *Medienverarbeitung in Java*. dpunkt.verlag, 2004.

[Fau93] Olivier Faugeras. *Three-Dimensional Computer Vision*. The MIT Press, 1993.

[Gro] Khronos Group. COLLADA - 3D Asset Exchange Schema, `http://khronos.org/collada/` (accessed on 10/17/2010).

[Gro08] COLLADA Work Group. COLLADA 1.4 Schema, `http://www.collada.org/2005/11/COLLADASchema/` (accessed on 10/17/2010), 2008.

[Her] Hugo Ortega Hernández. JCamCalib: A Camera Calibration Utility. v0.7, `http://dali.mty.itesm.mx/~hugo/thesis/JCamCalib/` (accessed on 10/22/2010).

[IMD] IMDB. Coraline (2009), `http://www.imdb.com/title/tt0327597/` (accessed on 10/19/2010).

[jMo] jMonkeyEngine.org. jME2 - jME wiki, `http://jmonkeyengine.org/wiki/doku.php/jme2` (accessed on 10/11/2010).

[Mat] The MathWorks. JAMA : A Java Matrix Package, `http://math.nist.gov/javanumerics/jama/` (accessed on 10/11/2010).

[oH] National Institutes of Health. ImageJ, `http://rsbweb.nih.gov/ij/` (accessed on 10/11/2010).

[Opta] OptiTrack. Camera Comparison Table, `http://www.naturalpoint.com/optitrack/products/camera-comparison.html` (accessed on 10/02/2010).

[Optb] OptiTrack. ARENA Tutorial Videos, `http://www.naturalpoint.com/optitrack/products/motion-capture/tutorials/arena/` (accessed on 10/02/2010).

[Ora] Oracle. Java Media Framework API (JMF), `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html` (accessed on 10/11/2010).

[Pis09] Hemerson Pistori. Hough Circles, `http://rsbweb.nih.gov/ij/plugins/hough-circles.html` (accessed on 10/02/2010), 2009.

[Rel09] Raz Public Relations. UC Merced Adopts VICON F40 Motion Capture System, `http://blog.digitalcontentproducer.com/briefingroom/2009/01/28/uc-merced-adopts-vicon-f40-motion-capture-system/` (accessed on 10/02/2010), 2009.

[Sta] Jens Stapelfeldt. Sensorgrößen für Digitalkameras berechnen, `http://www.bildergalerie-hamburg.de/foto-info/sensorgroesse-digitalkamera.php` (accessed on 10/02/2010).

[Tip07] Paul A. Tipler. *Physik für Wissenschaftler und Ingenieure*. Spektrum, 2007.

[Tsa86] Roger Y. Tsai. *An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.

Declaration of Authorship

I hereby declare that this thesis and the work presented in it was written within the registered timeframe and is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Cologne, 10/25/2010

_____

(Sven Bambach)